

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

Институт цифровых интеллектуальных систем  
Кафедра компьютерных систем управления

Направление подготовки: 15.03.04 Автоматизация технологических процессов и производств

**Отчет по лабораторной работе №2**

**по дисциплине «Программирование и Алгоритмизация»**

**Тема: «**Разработка простейшего графического приложения, реагирующего на действия пользователя**»**

Выполнил:

студент гр. АДБ-22-06 \_\_\_\_\_\_\_\_\_\_\_ Голубев С.Г.

(дата) (подпись)

Принял:

к.т.н. доцент \_\_\_\_\_\_\_\_\_\_\_ Пушков Р.Л.

(дата) (подпись)

**Москва 2023**

Оглавление

[Описание лабораторной работы: 3](#_Toc149313230)

[Создание проекта 3](#_Toc149313231)

[Создание простейшего приложения 4](#_Toc149313232)

[Добавление отрисовки графики 6](#_Toc149313233)

[Добавление динамики 8](#_Toc149313234)

[Добавление дополнительных препятствий 11](#_Toc149313235)

[Движение препятствия в качестве реакции на действия пользователя. 15](#_Toc149313236)

[Объяснение расчетов ограничений в методе CBall::Move 17](#_Toc149313237)

[Индивидуальное задание 3 вариант 18](#_Toc149313238)

[Вывод 21](#_Toc149313239)

[Приложения 22](#_Toc149313240)

[Приложение 1. Файл lab2.cpp. 22](#_Toc149313241)

[Приложение 2. CBall.h. 25](#_Toc149313242)

[Приложение 3. CBall.cpp 26](#_Toc149313243)

[Приложение 4. CHLimiter.h 28](#_Toc149313244)

[Приложение 5. CHLimiter.cpp 29](#_Toc149313245)

[Приложение 6. CVLimiter.h 30](#_Toc149313246)

[Приложение 7. CVLimiter.cpp 31](#_Toc149313247)

# Описание лабораторной работы:

**Цель лабораторной работы**: разработать простейшее графическое приложение, которое реагирует на действия пользователя.

## Создание проекта

1. Создаём новый проект в visual studio 2022. Для разработки графического приложения выбираем “Классическое приложение windows”. Далее назначаем имя проекту “lab2”.

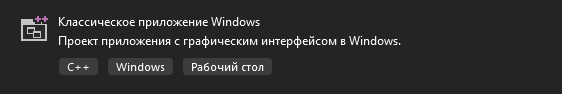
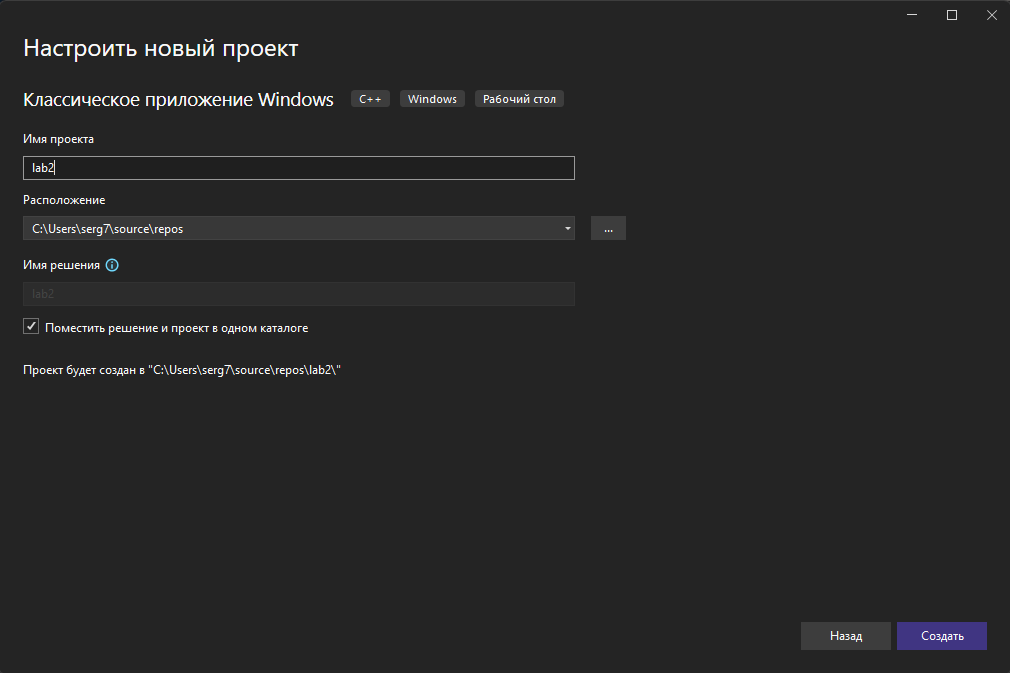


Рис. 1. Выбор шаблона “Классическое приложение Windows” Рис. 2. Настройка шаблона “Классическое приложение Windows”

## Создание простейшего приложения

1. Создаём точку входа для приложения “WinMain”, и подключаем заголовочный файл “windows.h”.

#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

return 0;

}

1. Добавляем в нашу программу реализацию оконного интерфейса: создаем класс окна приложения функцией “InitAppClass()”.

BOOL InitAppClass() {

ATOM class\_id;

WNDCLASS wc;

memset(&wc, 0, sizeof(wc));

wc.lpszMenuName = NULL;

wc.style = CS\_HREDRAW | CS\_VREDRAW;

wc.lpfnWndProc = (WNDPROC)WndProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hInstance = g\_hInstance;

wc.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wc.lpszClassName = L"lab 2";

class\_id = RegisterClass(&wc);

if (class\_id != 0)

return TRUE;

return FALSE;

}

1. Определяем следующие переменные глобально

HINSTANCE g\_hInstance;

int g\_nCmdShow;

HWND g\_mainWnd;

1. Заполняем эти переменные в запуске WinMain:

g\_hInstance = hInstance;

g\_nCmdShow = nCmdShow;

1. Создаем прототип функции, которая будет обрабатывать оконные сообщения.

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

1. Добавим функцию, отвечающую за создание окна:

BOOL InitWindow() {

g\_mainWnd = CreateWindow(L"lab2", L"Лабораторная работа №2",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

400, 400,

0, 0,

g\_hInstance,

0);

if (!g\_mainWnd) return FALSE;

ShowWindow(g\_mainWnd, g\_nCmdShow);

UpdateWindow(g\_mainWnd);

return TRUE;

}

1. Следующим шагом реализуем цикл обработки сообщений:

WPARAM StartMessageLoop() {

MSG msg;

while (GetMessage(&msg, 0, 0, 0))

DispatchMessage(&msg);

return msg.wParam;

1. Создаем функцию обработки сообщений

LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {

switch (msg) {

case WM\_DESTROY:

{

PostQuitMessage(0);

return 0;

}

default:

break;

}

return DefWindowProc(hWnd, msg, wParam, lParam);

}

1. Дополняем WinMain вызовами функций:

if (!InitAppClass())

return 0;

if (!InitWindow())

return 0;

return StartMessageLoop();

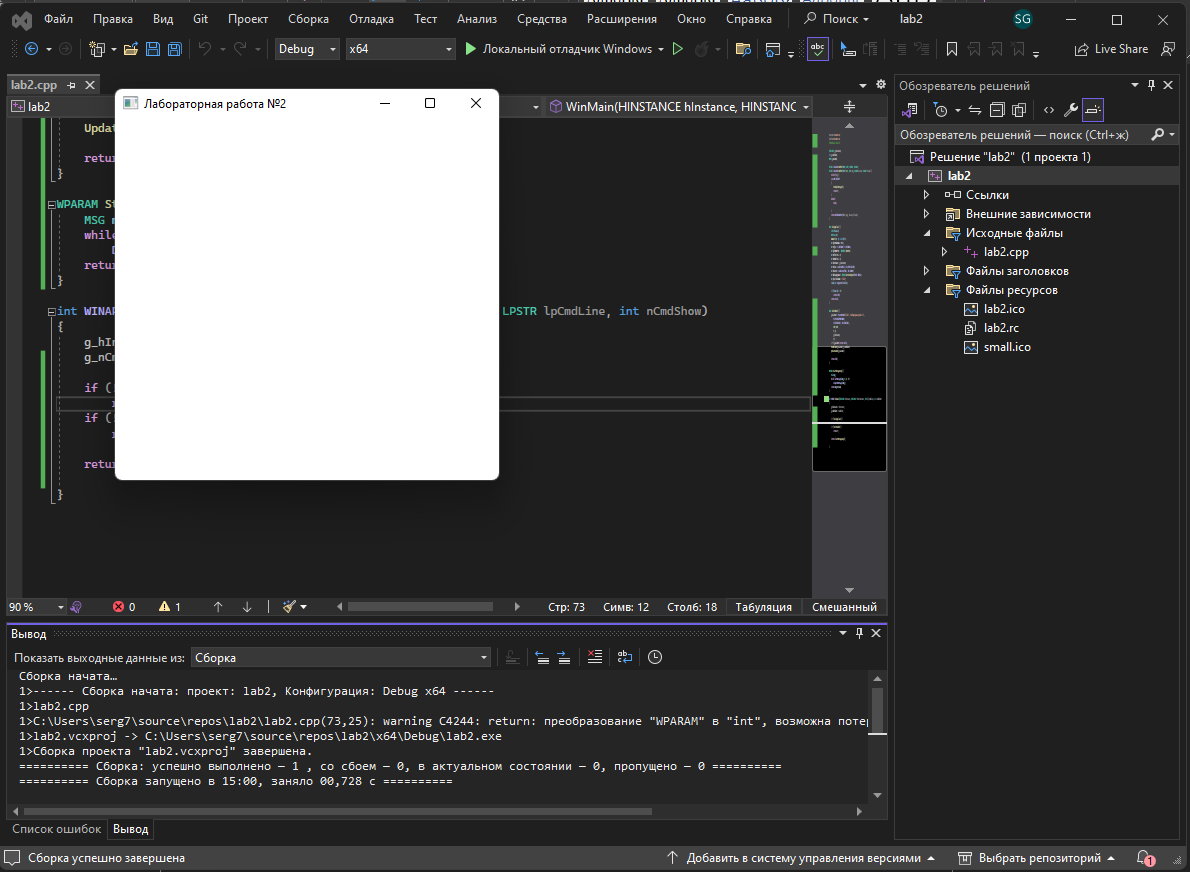
1. После компиляции программы наблюдаем окно:

Рис. 3. Вывод окна приложения

## Добавление отрисовки графики

1. Добавляем прототип функции Ellipse

BOOL Ellipse(

HDC hdc,

int nLeftrect,

int nTopRect,

int nRightRect,

int nBottomRect

);

1. Для отрисовки круга добавляем обработку сообщений WM\_PAINT в LRESULT CALLBACK WndProc

case WM\_PAINT: {

HDC hDC;

PAINTSTRUCT ps;

hDC = BeginPaint(hWnd, &ps);

Ellipse(hDC, 10, 10, 100, 100);

EndPaint(hWnd, &ps);

return 0;

}

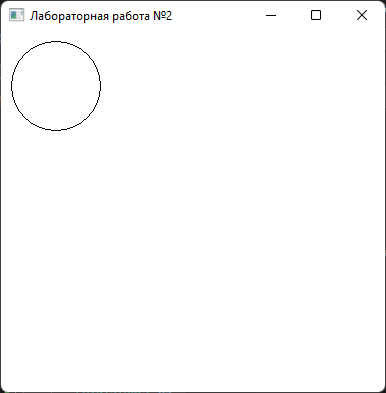
1. Получаем следующий результат работы программы: 

Рис. 4. Отрисовка круга

1. Реализуем шар в виде класса CBall, который разместим в файле CBall.cpp, а определение класса – в файле Cball.h

Реализация Cball.h:

#include<windows.h>

#include<windowsx.h>

class CBall {

HBRUSH hBrush;

public:

CBall();

~CBall();

void Draw(HDC dc);

};

Определение класса:

#include "CBall.h"

CBall::CBall() {

hBrush = CreateSolidBrush(RGB(0, 255, 0));

}

CBall::~CBall() {

DeleteObject(hBrush);

}

void CBall::Draw(HDC dc) {

HBRUSH hOldBrush;

hOldBrush = SelectBrush(dc, hBrush);

Ellipse(dc, 10, 10, 30, 30);

SelectBrush(dc, hOldBrush);

}

1. Добавим шарик в описание глобальных переменных

CBall b1;

1. Изменим обработчик WM\_PAINT:

case WM\_PAINT: {

HDC hDC;

PAINTSTRUCT ps;

hDC = BeginPaint(hWnd, &ps);

b1.Draw(hDC);

EndPaint(hWnd, &ps);

return 0;

}

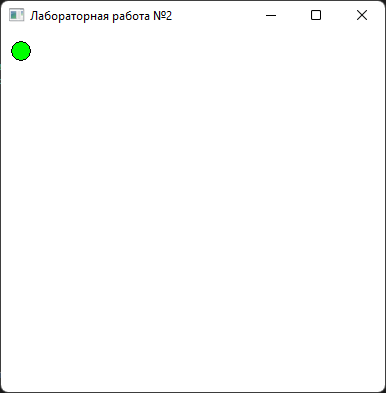
1. Получаем следующий результат выполнения программы: 

Рис. 5. Отображение шарика

## Добавление динамики

1. Для этого изменяем класс CBall в заголовочном файле

class CBall {

HBRUSH hBrush;

DWORD prev\_ticks;

double x, y;

double vx, vy;

double r;

RECT bounds;

public:

CBall(double x, double y, double vx, double vy, double r);

~CBall();

void Draw(HDC dc);

void Move(DWORD ticks);

};

1. Модифицируем функции класса в файле “CBall.cpp”

#include "CBall.h"

CBall::CBall(double x, double y, double vx, double vy, double r) {

hBrush = CreateSolidBrush(RGB(0, 255, 0));

this->x = x;

this->y = y;

this->vx = vx;

this->vy = vy;

this->r = r;

this->prev\_ticks = GetTickCount();

}

CBall::~CBall() {

DeleteObject(hBrush);

}

void CBall::Draw(HDC dc) {

HBRUSH hOldBrush;

hOldBrush = SelectBrush(dc, hBrush);

Ellipse(dc, x - r, y - r, x + r, y + r);

SelectBrush(dc, hOldBrush);

}

void CBall::Move(DWORD ticks) {

double s\_delta = ((double)(ticks - this->prev\_ticks)) / 1000.0;

this->prev\_ticks = ticks;

double dx = vx \* s\_delta;

double dy = vy \* s\_delta;

this->x += dx;

this->y += dy;

}

1. Вносим изменения в основной код. Изменяем определение глобальной переменной типа CBall

CBall b1(10,10,5,5,5);

1. Перерабатываем цикл обработки сообщений, меняя GetMessage на peekMessage.

WPARAM StartMessageLoop() {

MSG msg;

while (1) {

if (PeekMessage(&msg, 0, 0, 0, PM\_REMOVE)) {

if (msg.message == WM\_QUIT)

break;

DispatchMessage(&msg);

}

else {

OnIdle();

}

}

return msg.wParam;

}

1. Теперь реализуем функции OnIdle для ситуаций, когда сообщений нет в очереди

void OnIdle() {

DWORD ticks = GetTickCount();

b1.Move(ticks);

InvalidateRect(g\_mainWnd, NULL, TRUE);

}

1. При запуске можно наблюдать эффект мерцания, для его устранения ограничим частоту обновления до 50 кадров в секунду с помощью команды Sleep(20)
2. Следующий шаг: реализация отталкивания шарика от стенок окна. Для этого добавим в класс поле RECT.

class CBall {

HBRUSH hBrush;

DWORD prev\_ticks;

double x, y;

double vx, vy;

double r;

RECT bounds;

public:

CBall(double x, double y, double vx, double vy, double r);

~CBall();

void Draw(HDC dc);

void Move(DWORD ticks);

void SetBounds(RECT bnds);

};

1. Добавим метод SetBounds в CBall.cpp:

void CBall::SetBounds(RECT bnds) {

this->bounds = bnds;

}

1. Модифицируем метод Move с учетом границ

void CBall::Move(DWORD ticks) {

double s\_delta = ((double)(ticks - this->prev\_ticks)) / 1000.0;

if ((this->x >= bounds.right - r) && (this->vx > 0))

this->vx = -(this->vx);

if ((this->x <= r) && (this->vx < 0))

this->vx = -(this->vx);

if ((this->y >= bounds.bottom - r) && (this->vy > 0))

this->vy = -(this->vy);

if ((this->y <= r) && (this->vy < 0))

this->vy = -(this->vy);

this->prev\_ticks = ticks;

double dx = vx \* s\_delta;

double dy = vy \* s\_delta;

this->x += dx;

this->y += dy;

}

1. Редактируем код основной программы

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

g\_hInstance = hInstance;

g\_nCmsShow = nCmdShow;

if (!InitAppClass())

return 0;

if (!InitWindow())

return 0;

RECT cr;

GetClientRect(g\_mainWnd, &cr);

b1.SetBounds(cr);

return StartMessageLoop();

}

1. Добавим обработку сообщения WM\_SIZE

case WM\_SIZE:

{

RECT rect;

rect.top = 0;

rect.left = 0;

rect.bottom = HIWORD(lParam);

rect.right = LOWORD(lParam);

b1.SetBounds(rect);

return 0;

}

1. Увеличим скорость и размер шарика

CBall b1(10, 10, 150, 150, 50);

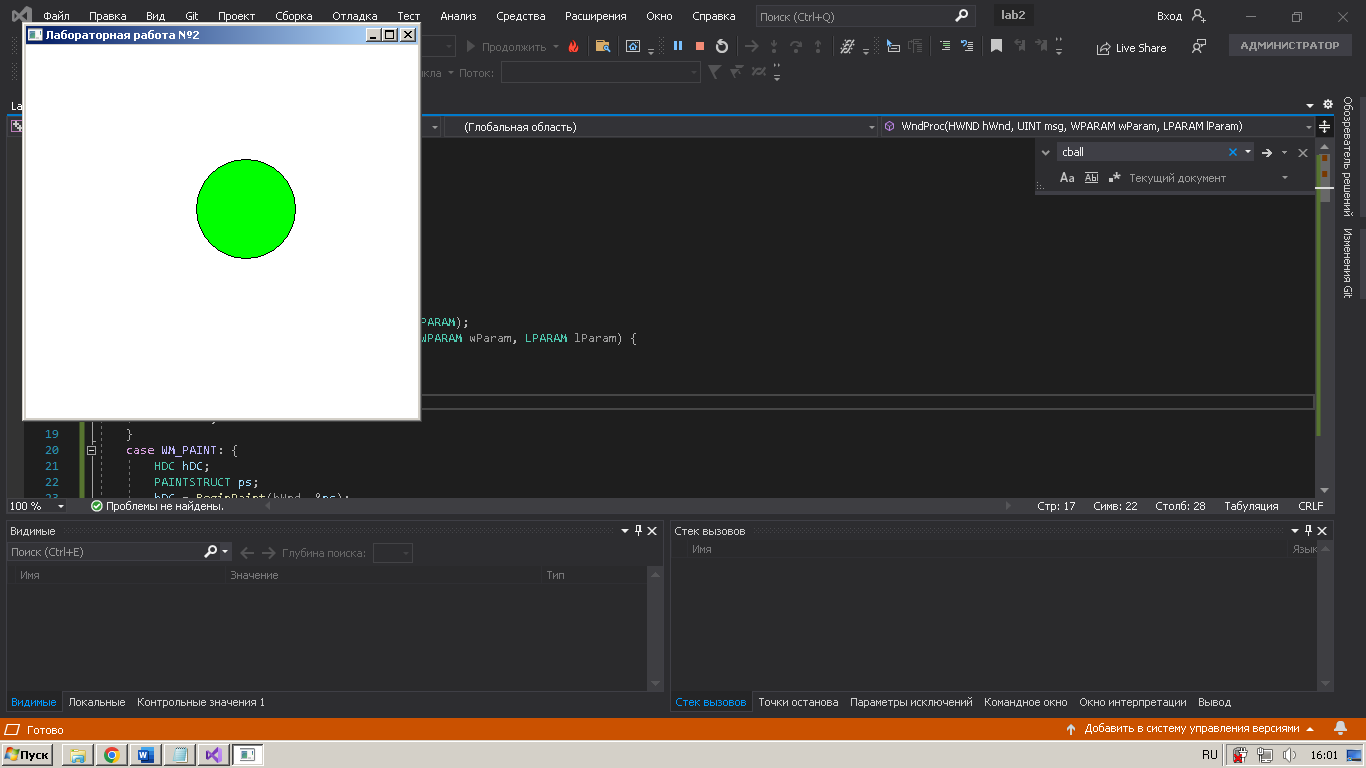


Рис. 6. Результат выполнения этапа “Добавление динамики”

## Добавление дополнительных препятствий

1. Создаем заголовочный файл препятствия CHLimiter.h

#pragma once

#include <windows.h>

#include <windowsx.h>

class CHLimiter

{

int y;

int xmin, xmax;

public:

CHLimiter(int xmin, int xmax, int y);

~CHLimiter();

void Draw(HDC dc);

int GetMaxX();

int GetMinX();

int GetY();

};

1. Создаем файл CHLimiter.cpp

#include "CHLimiter.h"

CHLimiter::CHLimiter(int xmin, int xmax, int y) {

this->xmax = xmax;

this->xmin = xmin;

this->y = y;

}

CHLimiter::~CHLimiter() {

}

void CHLimiter::Draw(HDC dc) {

::MoveToEx(dc, this->xmin, this->y, NULL);

::LineTo(dc, this->xmax, this->y);

}

int CHLimiter::GetMaxX() {

return this->xmax;

}

int CHLimiter::GetMinX() {

return this->xmin;

}

int CHLimiter::GetY() {

return this->y;

}

1. Вносим изменения в основной файл программы

#include <windows.h>

#include<windowsx.h>

#include "CBall.h"

#include "CHLimiter.h"

HINSTANCE g\_hInstance;

int g\_nCmdShow;

HWND g\_mainWnd;

CHLimiter h1(250, 350, 300);

CBall b1(10, 10, 150, 150, 50);

1. Добавление отрисовки объекту:

case WM\_PAINT: {

HDC hDC;

PAINTSTRUCT ps;

hDC = BeginPaint(hWnd, &ps);

b1.Draw(hDC);

h1.Draw(hDC)l;

EndPaint(hWnd, &ps);

return 0;

}

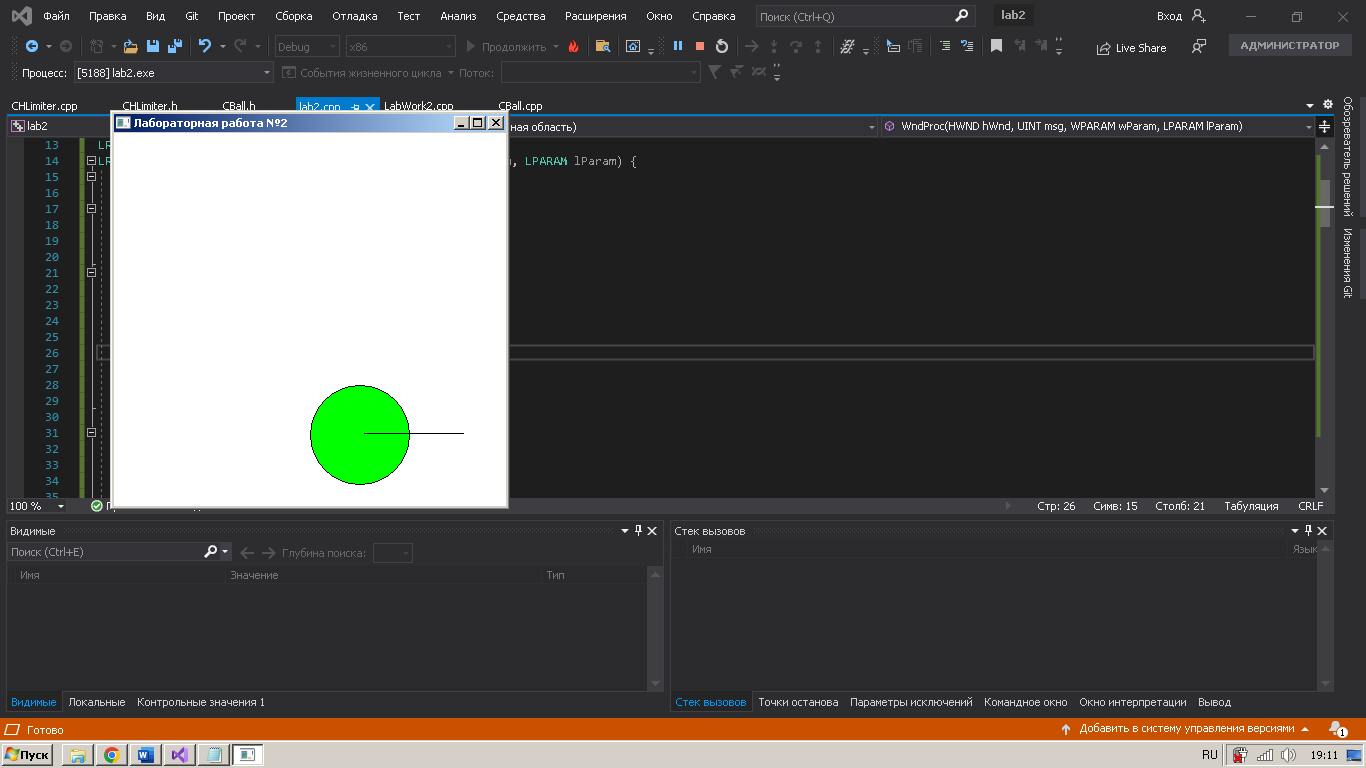


Рис. 7. Появление препятствия

1. Добавим в класс шарика параметр – указатель на объект границы и метод передачи границы:

#include <windows.h>

class CBall {

HBRUSH hBrush;

DWORD prev\_ticks;

double x, y;

double vx, vy;

double r;

RECT bounds;

CHLimiter\* limit1;

public:

CBall(double x, double y, double vx, double vy, double r);

~CBall();

void Draw(HDC dc);

void Move(DWORD ticks);

void SetBounds(RECT bnds);

void SetHLimiter(CHLimiter \* l);

};

1. Определим реализацию метода SetHLimiter

void CBall::SetHLimiter(CHLimiter\* l) {

this->limit1 = l;

}

1. Модифицируем код приложения:

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

g\_hInstance = hInstance;

g\_nCmdShow = nCmdShow;

if (!InitAppClass())

return 0;

if (!InitWindow())

return 0;

RECT cr;

GetClientRect(g\_mainWnd, &cr);

b1.SetBounds(cr);

b1.SetHLimiter(&h1);

return StartMessageLoop();

}

1. Модифицируем метод Move класс CBall

void CBall::Move(DWORD ticks) {

double s\_delta = ((double)(ticks - this->prev\_ticks)) / 1000.0;

if ((this->x >= bounds.right - r) && (this->vx > 0))

this->vx = -(this->vx);

if ((this->x <= r) && (this->vx < 0))

this->vx = -(this->vx);

if ((this->y >= bounds.bottom - r) && (this->vy > 0))

this->vy = -(this->vy);

if ((this->y <= r) && (this->vy < 0))

this->vy = -(this->vy);

if (((this->x + r) >= limit1->GetMinX()) && (this->x - r) <= limit1->GetMaxX())

{

if (this->vy > 0) {

if (((limit1->GetY() - this->y) < this->r) && ((limit1->GetY() - this->y) > 0))

this->vy = -(this->vy);

}

else {

if (((this->y - limit1->GetY()) < this->r) && ((this->y) - limit1->GetY()) > 0)

this->vy = -(this->vy);

}

}

this->prev\_ticks = ticks;

double dx = vx \* s\_delta;

double dy = vy \* s\_delta;

this->x += dx;

this->y += dy;

}

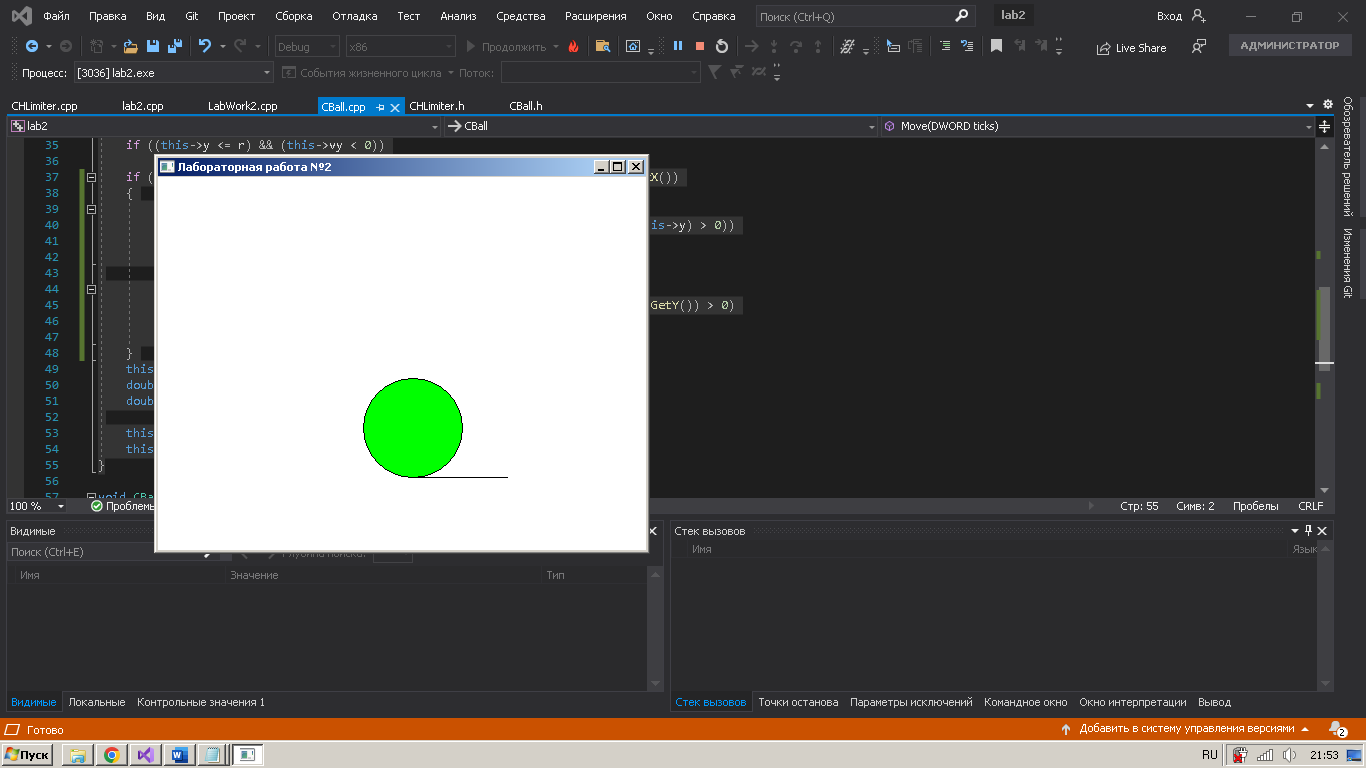


Рис. 8. Реализация коллизии

## Движение препятствия в качестве реакции на действия пользователя.

1. Первым делом добавим методы MoveX и MoveY для препятствия:

#pragma once

#include <windows.h>

#include <windowsx.h>

class CHLimiter

{

int y;

int xmin, xmax;

public:

CHLimiter(int xmin, int xmax, int y);

~CHLimiter();

void Draw(HDC dc);

int GetMaxX();

int GetMinX();

int GetY();

void MoveX(int inc);

void MoveY(int inc);

};

1. Реализуем эти два метода:

void CHLimiter::MoveX(int inc) {

this->xmax -= inc;

this->xmin -= inc;

}

void CHLimiter::MoveY(int inc) {

this->y -= inc;

}

1. Добавим функцию обработки нажатия клавиш

case WM\_KEYDOWN:

{

switch (wParam) {

case VK\_DOWN:

h1.MoveY(-5);

return 0;

case VK\_UP:

h1.MoveY(5);

return 0;

case VK\_LEFT:

h1.MoveX(5);

return 0;

case VK\_RIGHT:

h1.MoveX(-5);

return 0;

}

break;

}

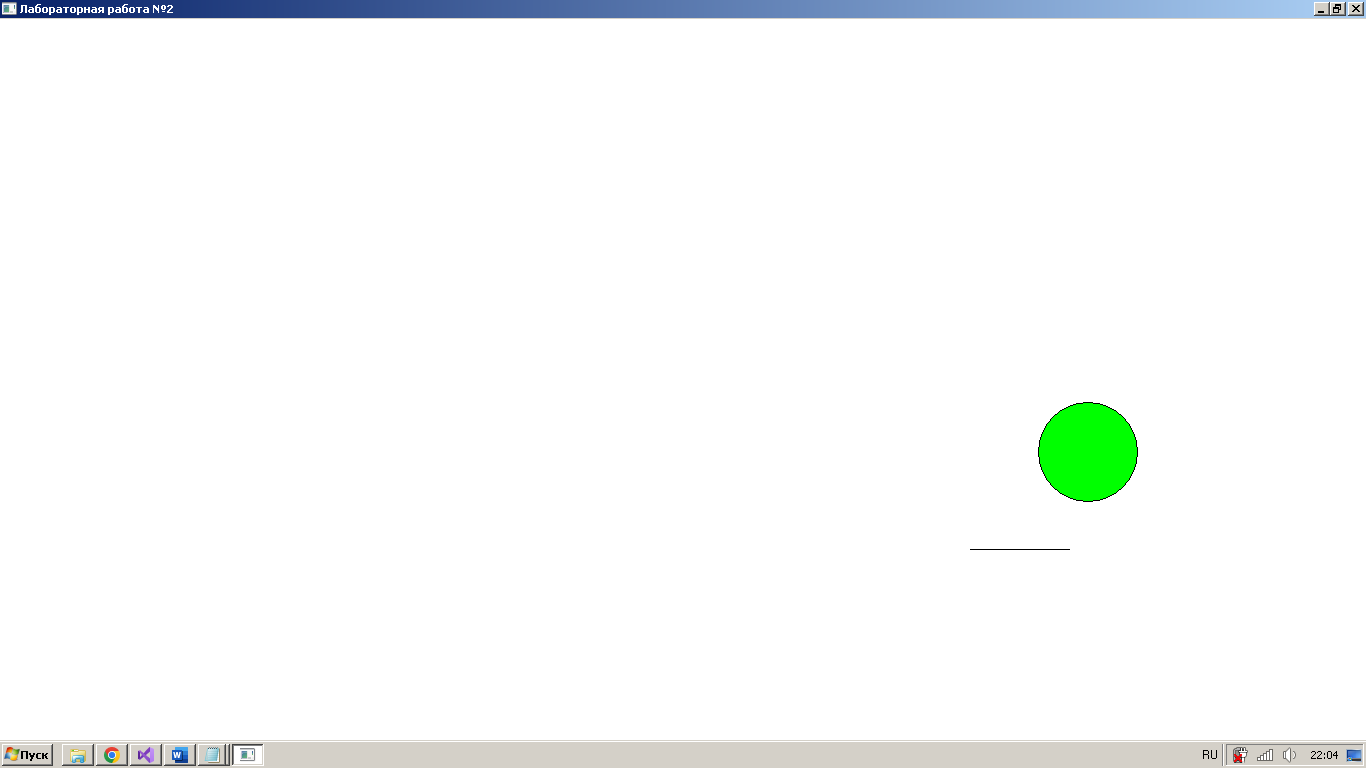


Рис. 9. Финальный результат работы программы

# Объяснение расчетов ограничений в методе CBall::Move

Первой строчкой

if ((this->x >= bounds.right - r) && (this->vx > 0))

this->vx = -(this->vx);

if ((this->x <= r) && (this->vx < 0))

this->vx = -(this->vx);

if ((this->y >= bounds.bottom - r) && (this->vy > 0))

this->vy = -(this->vy);

if ((this->y <= r) && (this->vy < 0))

this->vy = -(this->vy);

Эти условия меняют направление движения шарика на противоположное в случае, когда мяч приближении к границам окна.

if (((this->x + r) >= limit1->GetMinX()) && (this->x - r) <= limit1->GetMaxX())

{

if (this->vy > 0) {

if (((limit1->GetY() - this->y) < this->r) && ((limit1->GetY() - this->y) > 0))

this->vy = -(this->vy);

}

else {

if (((this->y - limit1->GetY()) < this->r) && ((this->y) - limit1->GetY()) > 0)

this->vy = -(this->vy);

}

}

Первое условие проверяет, находится ли мяч в пределах x координат, определенных ограничителем. Если да, то далее проверяется расстояние между y координатой мяча и y координатой препятствия. Если это расстояние меньше радиуса мяча и мяч движется в том направлении, где он должен пересекать ограничитель, его вертикальная скорость инвертируется.

# Индивидуальное задание 3 вариант

Реализуйте возможность добавления вертикального подвижного препятствия. Модифицируйте приложение, чтобы было 1 подвижное вертикальное препятствие и неподвижное горизонтальное.

1. Добавим два новых файла CVLimiter.h и CVLimiter.cpp в наш проект
2. Повторяем файлы CHLimiter.h и CHLimiter.cpp. Меняем местами X и Y
3. CVLimiter.h:

#pragma once

#include <windows.h>

class CVLimiter {

int x;

int ymin, ymax;

public:

CVLimiter(int x, int ymin, int ymax);

~CVLimiter();

void Draw(HDC dc);

int GetMinY();

int GetMaxY();

int GetX();

void MoveY(int inc);

};

1. CVLimiter.cpp:

#include "CVLimiter.h"

CVLimiter::CVLimiter(int x, int ymin, int ymax) {

this->x = x;

this->ymin = ymin;

this->ymax = ymax;

}

CVLimiter::~CVLimiter() {

}

void CVLimiter::Draw(HDC dc) {

::MoveToEx(dc, this->x, this->ymin, NULL);

::LineTo(dc, this->x, this->ymax);

}

int CVLimiter::GetMinY() {

return this->ymin;

}

int CVLimiter::GetMaxY() {

return this->ymax;

}

int CVLimiter::GetX() {

return this->x;

}

void CVLimiter::MoveY(int inc) {

this->ymin += inc;

this->ymax += inc;

}

1. Добавим вертикальное препятствие в CBall.h:

#include <windows.h>

#include "CHLimiter.h"

#include "CVLimiter.h" // подключение

class CBall {

HBRUSH hBrush;

DWORD prev\_ticks;

double x, y;

double vx, vy;

double r;

RECT bounds;

CHLimiter\* limit1;

CVLimiter\* limit2; // указатель

public:

CBall(double x, double y, double vx, double vy, double r);

~CBall();

void Draw(HDC dc);

void Move(DWORD ticks);

void SetBounds(RECT bnds);

void SetHLimiter(CHLimiter\* l);

void SetVLimiter(CVLimiter\* l); // функция

};

1. Изменения в CBall.cpp:

CBall::Move добавление коллизии:

if (((this->y + r) >= limit2->GetMinY()) && ((this->y - r) <= limit2->GetMaxY())) {

if (this->vx > 0) {

if (((limit2->GetX() - this->x) < this->r) && ((limit2->GetX() - this->x) > 0))

this->vx = -(this->vx);

}

else {

if (((this->x - limit2->GetX()) < this->r) && ((this->x - limit2->GetX()) > 0))

this->vx = -(this->vx);

}

}

void CBall::SetVLimiter(CVLimiter\* l) {

this->limit2 = l;

}

1. Изменения в lab2.cpp:

#include "CVLimiter.h"

CVLimiter v1(250, 350, 500);

v1.Draw(hDC);

case WM\_KEYDOWN:

{

switch (wParam) {

case VK\_DOWN:

v1.MoveY(5);

return 0;

case VK\_UP:

v1.MoveY(-5);

return 0;

case VK\_LEFT:

v1.MoveX(5);

return 0;

case VK\_RIGHT:

v1.MoveX(-5);

return 0;

}

b1.SetVLimiter(&v1);



Рис. 10. Результат работы программы

# Вывод

В ходе выполнения данной лабораторной работы мы получили навыки разработки простейшего графического приложения, которое реагирует на действия пользователя.

# Приложения

## Приложение 1. Файл lab2.cpp.

#include <windows.h>//заголовочный файл, в котором находится описание основных функций Windows API

#include<windowsx.h>

#include "CBall.h" //пределения класса CBall

#include "CHLimiter.h"

#include "CVLimiter.h"

HINSTANCE g\_hInstance; //это дескриптор экземпляра или дескриптор модуля. Операционная система использует это значение для идентификации исполняемого файла или EXE - файла при загрузке в память

int g\_nCmdShow;//это флаг, указывающий, является ли основное окно приложения свернуто, развернуто или отображается в обычном режиме.

HWND g\_mainWnd;//для определения дескриптора главного кода

CBall b1(10, 10, 150, 150, 50);//создание объекта типа CBall с именем b1 и передача конструктору координат, размеров и скорости

CHLimiter h1(250, 350, 300);//горизонтальное препятствие

CVLimiter v1(250, 350, 500);//верт. препятствие

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); //Объявление прототипа функции WndProc, которая является обработчиком сообщений для главного окна

LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {

switch (msg) {

case WM\_DESTROY://Если окно уничтожается

{

PostQuitMessage(0);

return 0;

}

case WM\_PAINT: {

HDC hDC;

PAINTSTRUCT ps;

hDC = BeginPaint(hWnd, &ps);

b1.Draw(hDC);

h1.Draw(hDC);

v1.Draw(hDC);

EndPaint(hWnd, &ps);

return 0;

}

case WM\_SIZE://При изменении размеров окна

{

RECT rect;

rect.top = 0;

rect.left = 0;

rect.bottom = HIWORD(lParam);

rect.right = LOWORD(lParam);

b1.SetBounds(rect);

return 0;

}

case WM\_KEYDOWN://обработка и генерация реакции на нажатие клавиш для препятствия

{

switch (wParam) {

case VK\_DOWN:

v1.MoveY(5);

return 0;

case VK\_UP:

v1.MoveY(-5);

return 0;

case VK\_LEFT:

v1.MoveX(5);

return 0;

case VK\_RIGHT:

v1.MoveX(-5);

return 0;

}

break;

}

default:

break;

}

return DefWindowProc(hWnd, msg, wParam, lParam);

}

BOOL InitAppClass() {// Определение функции, которая инициализирует класс приложения. Она создает и регистрирует класс окна с помощью структуры WNDCLASS и функций LoadIcon(), LoadCursor(), GetStockObject() и RegisterClass()

ATOM class\_id;

WNDCLASS wc;

memset(&wc, 0, sizeof(wc));

wc.lpszMenuName = NULL;

wc.style = CS\_HREDRAW | CS\_VREDRAW;

wc.lpfnWndProc = (WNDPROC)WndProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hInstance = g\_hInstance;

wc.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wc.lpszClassName = L"lab2";

class\_id = RegisterClass(&wc);

if (class\_id != 0)

return TRUE;

return FALSE;

}

BOOL InitWindow() { // Определение функции InitWindow(), которая инициализирует главное окно, используя функцию CreateWindow().Она устанавливает различные параметры окна, такие как заголовок, стиль, расположение и размеры.Затем окно отображается с помощью ShowWindow() и обновляется с помощью UpdateWindow()

g\_mainWnd = CreateWindow(L"lab2", L"Лабораторная работа №2",

WS\_OVERLAPPEDWINDOW, //стиль окна: перекрывающееся окно

CW\_USEDEFAULT, CW\_USEDEFAULT,

400, 400,

0, 0, //родительское окно и дескриптор меню

g\_hInstance, //дескриптор экземпляра

0);

if (!g\_mainWnd) return FALSE;

ShowWindow(g\_mainWnd, g\_nCmdShow);

UpdateWindow(g\_mainWnd);

return TRUE;

}

void OnIdle() { //вызываться тогда, когда сообщений в очереди нет

DWORD ticks = GetTickCount();

b1.Move(ticks);

InvalidateRect(g\_mainWnd, NULL, TRUE);

}

WPARAM StartMessageLoop() { // цикл обработки сообщений

MSG msg;

while (1) {

if (PeekMessage(&msg, 0, 0, 0, PM\_REMOVE)) {

if (msg.message == WM\_QUIT)

break;

DispatchMessage(&msg);

}

else {

Sleep(20);

OnIdle();

}

}

return msg.wParam;

}

//BOOL Ellipse//Объявление прототипа функции Ellipse, которая рисует эллипс на указанном контексте рисования (hdc) внутри описывающего прямоугольника, определенного четырьмя координатами

// HDC hdc,

// int nLeftrect,

// int nTopRect,

// int nRightRect,

// int nBottomRect

//);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) { //Точка входа приложения

g\_hInstance = hInstance;

g\_nCmdShow = nCmdShow;

if (!InitAppClass())

return 0;

if (!InitWindow())

return 0;

RECT cr;

GetClientRect(g\_mainWnd, &cr);

b1.SetBounds(cr);

b1.SetHLimiter(&h1);

b1.SetVLimiter(&v1);

return StartMessageLoop();

}

## Приложение 2. CBall.h.

#pragma once //препроцессорная директива нужная для того, чтобы исходный файл при компиляции подключался строго один раз

#include <windows.h>

#include "CHLimiter.h"

#include "CVLimiter.h"

class CBall {

HBRUSH hBrush;

DWORD prev\_ticks;

double x, y;

double vx, vy; //скорость

double r;

RECT bounds;

CHLimiter\* limit1;

CVLimiter\* limit2;

public:

CBall(double x, double y, double vx, double vy, double r);

~CBall();

void Draw(HDC dc);

void Move(DWORD ticks);//в методе Move реализация отталкивания шарика от стенок окна

void SetBounds(RECT bnds);

void SetHLimiter(CHLimiter\* l);

void SetVLimiter(CVLimiter\* l);

};

## Приложение 3. CBall.cpp

#include "CBall.h"

#include <windows.h>

#include <wingdi.h>

CBall::CBall(double x, double y, double vx, double vy, double r) {

hBrush = CreateSolidBrush(RGB(0, 255, 0));

this->x = x;

this->y = y;

this->vx = vx;

this->vy = vy;

this->r = r;

this->prev\_ticks = GetTickCount();

}

CBall::~CBall() {

DeleteObject(hBrush);

}

void CBall::Draw(HDC dc) {

HBRUSH hOldBrush = (HBRUSH)SelectObject(dc, hBrush);

Ellipse(dc, x - r, y - r, x + r, y + r);

SelectObject(dc, hOldBrush);

}

void CBall::Move(DWORD ticks) {//определение метода Move класса CBall, который принимает значение времени в миллисекундах ticks. Рассчитывается время между движениями в секундах. Выполняется проверка и изменение направления движения, если шарик достигает границ полей или ограничителей. Рассчитываются приращения координат x и y, и они добавляются к текущим координатам шарика.

double s\_delta = ((double)(ticks - this->prev\_ticks)) / 1000.0;

if ((this->x >= bounds.right - r) && (this->vx > 0))

this->vx = -(this->vx);

if ((this->x <= r) && (this->vx < 0))

this->vx = -(this->vx);

if ((this->y >= bounds.bottom - r) && (this->vy > 0))

this->vy = -(this->vy);

if ((this->y <= r) && (this->vy < 0))

this->vy = -(this->vy);

if (((this->x + r) >= limit1->GetMinX()) && (this->x - r) <= limit1->GetMaxX())

{

if (this->vy > 0) {

if (((limit1->GetY() - this->y) < this->r) && ((limit1->GetY() - this->y) > 0))

this->vy = -(this->vy);

}

else {

if (((this->y - limit1->GetY()) < this->r) && ((this->y) - limit1->GetY()) > 0)

this->vy = -(this->vy);

}

}

if (((this->y + r) >= limit2->GetMinY()) && ((this->y - r) <= limit2->GetMaxY())) {

if (this->vx > 0) {

if (((limit2->GetX() - this->x) < this->r) && ((limit2->GetX() - this->x) > 0))

this->vx = -(this->vx);

}

else {

if (((this->x - limit2->GetX()) < this->r) && ((this->x - limit2->GetX()) > 0))

this->vx = -(this->vx);

}

}

this->prev\_ticks = ticks;

double dx = vx \* s\_delta;

double dy = vy \* s\_delta;

this->x += dx;

this->y += dy;

}

void CBall::SetBounds(RECT bnds) {

this->bounds = bnds;

}

void CBall::SetHLimiter(CHLimiter\* l) {//определение метода SetHLimiter класса CBall, который принимает указатель на объект CHLimiter l.

this->limit1 = l;

}

void CBall::SetVLimiter(CVLimiter\* l) {

this->limit2 = l;

}

## Приложение 4. CHLimiter.h

#pragma once

#include <windows.h>

#include <windowsx.h>

class CHLimiter

{

int y;

int xmin, xmax;

public:

CHLimiter(int xmin, int xmax, int y);

~CHLimiter();

void Draw(HDC dc);

int GetMaxX();

int GetMinX();

int GetY();

void MoveX(int inc);

void MoveY(int inc);

};

## Приложение 5. CHLimiter.cpp

#include "CHLimiter.h"

CHLimiter::CHLimiter(int xmin, int xmax, int y) {

this->xmax = xmax;

this->xmin = xmin;

this->y = y;

}

CHLimiter::~CHLimiter() {

}

void CHLimiter::Draw(HDC dc) {

::MoveToEx(dc, this->xmin, this->y, NULL);

::LineTo(dc, this->xmax, this->y);

}

void CHLimiter::MoveX(int inc) {

this->xmax -= inc;

this->xmin -= inc;

}

void CHLimiter::MoveY(int inc) {

this->y -= inc;

}

int CHLimiter::GetMaxX() {

return this->xmax;

}

int CHLimiter::GetMinX() {

return this->xmin;

}

int CHLimiter::GetY() {

return this->y;

}

## Приложение 6. CVLimiter.h

#pragma once

#include <windows.h>

class CVLimiter {

int x;

int ymin, ymax;

public:

CVLimiter(int x, int ymin, int ymax);

~CVLimiter();

void Draw(HDC dc);

int GetMinY();

int GetMaxY();

int GetX();

void MoveY(int inc);

void MoveX(int inc);

};

## Приложение 7. CVLimiter.cpp

#include "CVLimiter.h"

CVLimiter::CVLimiter(int x, int ymin, int ymax) {

this->x = x;

this->ymin = ymin;

this->ymax = ymax;

}

CVLimiter::~CVLimiter() {

}

void CVLimiter::Draw(HDC dc) {

::MoveToEx(dc, this->x, this->ymin, NULL);

::LineTo(dc, this->x, this->ymax);

}

int CVLimiter::GetMinY() {

return this->ymin;

}

int CVLimiter::GetMaxY() {

return this->ymax;

}

int CVLimiter::GetX() {

return this->x;

}

void CVLimiter::MoveY(int inc) {

this->ymin += inc;

this->ymax += inc;

}

void CVLimiter::MoveX(int inc) {

this->x -= inc;

}