Листинг программы

#include "windows.h"

#include "resource.h"

#include "random.h"

#include <list>

#include <queue>

#include <stack>

#include <strstream>

#include <tchar.h>

#define MENU\_HEIGHT 45

#define TOP\_HEIGHT 46 // Высота табло (где отображаются очки) в пикселах

#define CELL\_SIZE 45 //Размер стороны ячейки в пикселах (они все квадратные)

#define N1 4 //Количество картинок для появления шарика

#define N2 12 //Количество картинок для прыжка шарика

#define N3 9 //Количество картинок для удаления шарика

#define MAX\_MAP\_X 20 //Максимальный размер поля по x

#define MAX\_MAP\_Y 12 //Максимальный размер поля по y

struct {

//Состояние автомата для этой ячейки

int y;

//Цвет шарика в этой ячейке

int color;

//Цвет шарика-подсказки в этой ячейке

//(когда игровой шарик проходит через эту ячейку, тогда color != pre\_color)

int pre\_color;

//Номер картинки, выводимой в текущий момент, при появлении, удалении, прыжке шарика

int num\_pic;

}

// Игровое поле

map[MAX\_MAP\_X][MAX\_MAP\_Y];

// Цвет игрового шарика

int ball\_color;

struct info

{

int score;

int time;

char name[30];

}

// Лучшие результаты

leaders[3];

// Минимальный результат, чтобы попасть в таблицу

const info null\_leader = { 100,3600,"noname" };

// Состояние автомата управления игрой

int y\_lines;

// Время игры

int gametime;

// Очки

int gamescore;

// Тип игры: 0-easy, 1-normal, 2-hard, 3-custom

int gametype;

// Текущий размер поля

int max\_x; // по x

int max\_y; // по y

// Количество появляющихся шаров

int app\_balls;

//Количество удаляемых шаров

int del\_balls;

TCHAR MY\_SOUND0[] = \_T("res//move.wav");

TCHAR MY\_SOUND1[] = \_T("res//destroy.wav");

TCHAR MY\_SOUND2[] = \_T("res//fetus.wav");

TCHAR MY\_SOUND3[] = \_T("res//cantmove.wav");

HANDLE bmp\_0, bmp\_prestand, bmp\_stand, bmp\_jump[N2], bmp\_explode[N3], bmp\_appear[N1], bmp\_numbers, bmp\_points;

HANDLE\* bmp[6];

HDC hDC;

HDC hCompatibleDC;

HWND hWnd;

HINSTANCE hInst;

TCHAR szTitle[] = "Lines";

TCHAR szWindowClass[] = "LINES";

RECT clRect;

// Класс "Ячейка"

class cell

{

public:

int posx; // Позиция ячейки на поле (слева направо от 0)

int posy; // Позиция ячейки на поле (сверху вниз от 0)

bool operator ==(const cell& b) const {

return (b.posx == posx && b.posy == posy);

}

bool operator !=(const cell& b) const {

return (!(\*this == b));

}

int& State() const {

return map[posx][posy].y;

}

int& Color() const {

return map[posx][posy].color;

}

int& PreColor() const {

return map[posx][posy].pre\_color;

}

int& NumPic() const {

return map[posx][posy].num\_pic;

}

// автомат "ячейка"

void ACell(int e) const

{

int& y = State();

switch (y)

{

// Отсутствие шара

case 0:

if (e == 1) {

z0();

y = 3;

}

if (e == 2 && x0()) {

z4();

y = 1;

}

else if (e == 2) {

z1();

y = 1;

}

break;

// Подсказка

case 1:

if (e == 1) {

z2();

y = 3;

}

if (e == 3) y = 2;

break;

// Появление шара

case 2:

if (e == 3 && x1()) {

z6();

y = 3;

}

else if (e == 3) {

z5();

}

break;

// Шар стоит

case 3:

if (e == 0 && x0()) {

z4();

y = 1;

}

else if (e == 0) {

z3();

y = 0;

}

if (e == 4) y = 5;

if (e == 5) y = 4;

break;

// Шар прыгает

case 4:

if (e == 0) {

z6();

z3();

y = 0;

}

if (e == 6) {

z6();

y = 3;

}

if (e == 5 && x2()) {

z6();

}

else if (e == 5) {

z5();

}

break;

// Удаление шара

case 5:

if (e == 4 && x3() && x0()) {

z6();

z4();

y = 1;

}

else if (e == 4 && x3()) {

z6();

z3();

y = 0;

}

else if (e == 4) {

z5();

}

break;

}

DrawState();

}

void DrawState() const

{

SelectObject(hCompatibleDC, bmp[State()][NumPic()]);

BitBlt(

hDC,

posx \* CELL\_SIZE,

TOP\_HEIGHT + posy \* CELL\_SIZE,

CELL\_SIZE, CELL\_SIZE,

hCompatibleDC,

Color() \* CELL\_SIZE,

0,

SRCCOPY);

}

private:

//ВХОДЯЩИЕ ПЕРЕМЕННЫЕ

//Требуется восстановление подсказки

bool x0() const

{

return PreColor() != -1;

}

//Последняя стадия появления шара

bool x1() const

{

return (NumPic() == N1 - 1);

}

//Последняя стадия прыжка для шара

bool x2() const

{

return (NumPic() == N2 - 1);

}

//Последняя стадия удаления шара

bool x3() const

{

return (NumPic() == N3 - 1);

}

//ВЫХОДЯЩИЕ ВОЗДЕЙСТВИЯ

//Установить в ячейке двигающийся шар

void z0() const

{

Color() = ball\_color;

}

//Генерировать подсказку

void z1() const

{

Color() = random(7);

}

//Очистить ячейку

void z2() const

{

PreColor() = Color(); Color() = ball\_color;

}

//Установить в ячейке с подсказкой двигающийся шар

void z3() const

{

Color() = 0;

}

//Восстановить подсказку

void z4() const

{

Color() = PreColor(); PreColor() = -1;

}

//Вывести на экран следующую картинку для текущего состояния

void z5() const

{

++NumPic();

}

//Вывести на экран первую картинку для текущего состояния

void z6() const

{

NumPic() = 0;

}

};

//Игровой шар

cell ball;

//Выбранный шар (ткнули мышкой)

cell click\_ball;

//Список появляющихся шаров

std::list<cell> appear\_list;

//Список удаляющихся шаров

std::list<cell> explode\_list;

//Путь по которому проходит двигающийся шар

std::stack<cell> path;

std::list<cell>::iterator itr;

// Прототипы функций, встречаемых далее в программе

void ALines(int);

void z0();

void z1\_1();

void z1\_2();

void z1\_3();

void z2\_1();

void z2\_2();

void z2\_3();

void z2\_4();

void z3\_1();

void z3\_2();

void z4\_1();

void z4\_2();

bool xk0();

bool xk1();

bool xk2();

bool x0();

bool x1();

bool x2();

bool x3();

bool x4();

bool x5();

bool FindEmptyCell(cell&);

void GenerateAppearList();

void CheckAppearList();

bool FindPath(const cell&, const cell&);

bool CheckLines(const cell&);

bool Valid(const cell&);

void GameOver();

void NewGame();

void DrawTime();

void DrawScore();

void DrawTop();

void CheckCustomParameters();

void GetInfo();

void WriteInfo();

ATOM MyRegisterClass(HINSTANCE);

BOOL InitInstance(HINSTANCE, int);

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

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

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

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

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

//Главная функция окна

int APIENTRY WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{

MSG msg;

HACCEL hAccelTable;

randomize();

hInst = hInstance;

MyRegisterClass(hInstance);

if (!InitInstance(hInstance, nCmdShow)) {

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC\_LINES);

//Главный цикл сообщений Windows

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

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return msg.wParam;

}

//Регистрация класса окна

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = (WNDPROC)WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInst, (LPCTSTR)IDI\_LINES);

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

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = (LPCSTR)IDC\_LINES;

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, (LPCTSTR)IDI\_SMALL);

return RegisterClassEx(&wcex);

}

// Создание и отображение окна

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {

hWnd = CreateWindow(

szWindowClass,

szTitle,

WS\_CAPTION | WS\_OVERLAPPED | WS\_MINIMIZEBOX | WS\_SYSMENU,

CW\_USEDEFAULT, 0, 1600, 600, NULL, NULL, hInstance, NULL);

if (!hWnd) {

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

// Главная функция окна

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

{

static PAINTSTRUCT ps;

static RECT Rect;

static HMENU hMenu;

static HKEY hKey;

static cell l;

static int temp;

static FILE\* f;

switch (message) {

// Обработка сообщения при создании окна

case WM\_CREATE:

randomize();

bmp[0] = &bmp\_0;

bmp[1] = &bmp\_prestand;

bmp[2] = &bmp\_appear[0];

bmp[3] = &bmp\_stand;

bmp[4] = &bmp\_jump[0];

bmp[5] = &bmp\_explode[0];

bmp\_0 = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_0));

bmp\_prestand = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_PRESTAND));

bmp\_appear[0] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_APPEAR\_1));

bmp\_appear[1] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_APPEAR\_2));

bmp\_appear[2] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_APPEAR\_3));

bmp\_appear[3] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_APPEAR\_4));

bmp\_stand = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_STAND));

bmp\_jump[0] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_3));

bmp\_jump[1] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_2));

bmp\_jump[2] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_1));

bmp\_jump[3] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_2));

bmp\_jump[4] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_3));

bmp\_jump[5] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_STAND));

bmp\_jump[6] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_4));

bmp\_jump[7] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_5));

bmp\_jump[8] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_6));

bmp\_jump[9] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_5));

bmp\_jump[10] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_JUMP\_4));

bmp\_jump[11] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_STAND));

bmp\_explode[0] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_1));

bmp\_explode[1] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_2));

bmp\_explode[2] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_3));

bmp\_explode[3] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_4));

bmp\_explode[4] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_5));

bmp\_explode[5] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_6));

bmp\_explode[6] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_7));

bmp\_explode[7] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_8));

bmp\_explode[8] = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_EXPLODE\_9));

bmp\_numbers = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_NUMBERS));

bmp\_points = LoadBitmap(hInst, MAKEINTRESOURCE(IDB\_POINTS));

hDC = GetDC(hWnd);

hCompatibleDC = CreateCompatibleDC(hDC);

GetInfo();

hMenu = GetSubMenu(GetMenu(hWnd), 0);

CheckMenuItem(hMenu, gametype + IDM\_EASY, MF\_CHECKED);

GetWindowRect(hWnd, &Rect);

MoveWindow(hWnd, Rect.left, Rect.top, (CELL\_SIZE + 1.666) \* max\_x, TOP\_HEIGHT + (CELL\_SIZE + 1.666) \* max\_y + MENU\_HEIGHT, TRUE);

NewGame();

SetTimer(hWnd, 0, 1000, NULL);

break;

// Обработка сообщения от нажатия левой кнопки мыши

case WM\_LBUTTONDOWN:

click\_ball.posx = LOWORD(lParam) / CELL\_SIZE;

click\_ball.posy = (HIWORD(lParam) - TOP\_HEIGHT) / 45;

ALines(0);

break;

// Обработка сообщения от таймера

case WM\_TIMER:

switch (LOWORD(wParam)) {

case 0:

gametime++;

DrawTime();

break;

case 1:

ALines(1);

break;

}

break;

// Обработка сообщения WM\_COMMAND

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case IDM\_EASY:

max\_x = 9;

max\_y = 6;

app\_balls = 2;

del\_balls = 4;

break;

case IDM\_NORMAL:

max\_x = 9;

max\_y = 9;

app\_balls = 3;

del\_balls = 5;

break;

case IDM\_HARD:

max\_x = 20;

max\_y = 12;

app\_balls = 10;

del\_balls = 4;

break;

case IDM\_CUSTOM:

DialogBox(hInst, (LPCTSTR)IDD\_CUSTOMBOX, hWnd, (DLGPROC)Custom);

CheckCustomParameters();

break;

}

switch (LOWORD(wParam)) {

case IDM\_ABOUT:

DialogBox(hInst, (LPCTSTR)IDD\_ABOUTBOX, hWnd, (DLGPROC)About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

case IDM\_EASY:

case IDM\_NORMAL:

case IDM\_HARD:

case IDM\_CUSTOM:

CheckMenuItem(hMenu, gametype + IDM\_EASY, MF\_UNCHECKED);

CheckMenuItem(hMenu, LOWORD(wParam), MF\_CHECKED);

gametype = LOWORD(wParam) - IDM\_EASY;

GetWindowRect(hWnd, &Rect);

MoveWindow(

hWnd,

Rect.left,

Rect.top,

(CELL\_SIZE + 1.666) \* max\_x,

TOP\_HEIGHT + (CELL\_SIZE + 1.666) \* max\_y + MENU\_HEIGHT,

TRUE);

case IDM\_NEW:

NewGame();

InvalidateRect(hWnd, NULL, FALSE);

break;

case IDM\_BESTRESULTS:

DialogBox(hInst, (LPCTSTR)IDD\_BESTRESULTSBOX, hWnd, (DLGPROC)BestResults);

break;

default:

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

}

break;

// Обработка сообщения при отрисовке окна

case WM\_PAINT:

BeginPaint(hWnd, &ps);

DrawTop();

for (l.posx = 0; l.posx < max\_x; l.posx++) {

for (l.posy = 0; l.posy < max\_y; l.posy++) {

l.DrawState();

}

}

EndPaint(hWnd, &ps);

break;

// Обработка сообщения при уничтожении окна

case WM\_DESTROY:

DeleteObject(bmp\_0);

DeleteObject(bmp\_prestand);

DeleteObject(bmp\_appear[0]);

DeleteObject(bmp\_appear[1]);

DeleteObject(bmp\_appear[2]);

DeleteObject(bmp\_appear[3]);

DeleteObject(bmp\_stand);

DeleteObject(bmp\_jump[0]);

DeleteObject(bmp\_jump[1]);

DeleteObject(bmp\_jump[2]);

DeleteObject(bmp\_jump[3]);

DeleteObject(bmp\_jump[4]);

DeleteObject(bmp\_jump[5]);

DeleteObject(bmp\_jump[6]);

DeleteObject(bmp\_jump[7]);

DeleteObject(bmp\_jump[8]);

DeleteObject(bmp\_jump[9]);

DeleteObject(bmp\_jump[10]);

DeleteObject(bmp\_jump[11]);

DeleteObject(bmp\_explode[0]);

DeleteObject(bmp\_explode[1]);

DeleteObject(bmp\_explode[2]);

DeleteObject(bmp\_explode[3]);

DeleteObject(bmp\_explode[4]);

DeleteObject(bmp\_explode[5]);

DeleteObject(bmp\_explode[6]);

DeleteObject(bmp\_explode[7]);

DeleteObject(bmp\_explode[8]);

DeleteObject(bmp\_numbers);

DeleteObject(bmp\_points);

DeleteDC(hCompatibleDC);

ReleaseDC(hWnd, hDC);

WriteInfo();

PostQuitMessage(0);

break;

// Обработка сообщения по умолчанию

default:

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

}

return 0;

}

// Главная функция окна диалога About

LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message) {

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL) {

EndDialog(hDlg, LOWORD(wParam));

return TRUE;

}

break;

}

return FALSE;

}

// Главная функция окна диалога Custom

LRESULT CALLBACK Custom(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static char szVal[10];

switch (message)

{

case WM\_INITDIALOG:

std::ostrstream(szVal, sizeof(szVal)) << max\_x << std::ends;

SendDlgItemMessage(hDlg, IDC\_EDIT1, WM\_SETTEXT, (WPARAM)0, (LPARAM)szVal);

std::ostrstream(szVal, sizeof(szVal)) << max\_y << std::ends;

SendDlgItemMessage(hDlg, IDC\_EDIT2, WM\_SETTEXT, (WPARAM)0, (LPARAM)szVal);

std::ostrstream(szVal, sizeof(szVal)) << app\_balls << std::ends;

SendDlgItemMessage(hDlg, IDC\_EDIT3, WM\_SETTEXT, (WPARAM)0, (LPARAM)szVal);

std::ostrstream(szVal, sizeof(szVal)) << del\_balls << std::ends;

SendDlgItemMessage(hDlg, IDC\_EDIT4, WM\_SETTEXT, (WPARAM)0, (LPARAM)szVal);

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK)

{

SendDlgItemMessage(hDlg, IDC\_EDIT1, WM\_GETTEXT, (WPARAM)sizeof(szVal), (LPARAM)szVal);

std::istrstream(szVal, sizeof(szVal)) >> max\_x;

SendDlgItemMessage(hDlg, IDC\_EDIT2, WM\_GETTEXT, (WPARAM)sizeof(szVal), (LPARAM)szVal);

std::istrstream(szVal, sizeof(szVal)) >> max\_y;

SendDlgItemMessage(hDlg, IDC\_EDIT3, WM\_GETTEXT, (WPARAM)sizeof(szVal), (LPARAM)szVal);

std::istrstream(szVal, sizeof(szVal)) >> app\_balls;

SendDlgItemMessage(hDlg, IDC\_EDIT4, WM\_GETTEXT, (WPARAM)sizeof(szVal), (LPARAM)szVal);

std::istrstream(szVal, sizeof(szVal)) >> del\_balls;

EndDialog(hDlg, LOWORD(wParam));

}

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return TRUE;

}

break;

}

return FALSE;

}

// Главная функция окна диалога BestResult

LRESULT CALLBACK BestResults(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

static char szVal[50];

static int i, h, m1, m2, s1, s2;

switch (message) {

case WM\_INITDIALOG:

for (i = 0; i < 3; i++) {

h = leaders[i].time;

s2 = h % 60; h /= 60;

s1 = s2 % 10; s2 /= 10;

m2 = h % 60; h /= 60;

m1 = m2 % 10; m2 /= 10;

std::ostrstream(szVal, sizeof(szVal)) << leaders[i].name << std::ends;;

SendDlgItemMessage(hDlg, IDC\_EDIT1 + i, WM\_SETTEXT, (WPARAM)0, (LPARAM)szVal);

std::ostrstream(szVal, sizeof(szVal)) << leaders[i].score << " " << h << ':' << m2 << m1 << ':' << s2 << s1 << std::ends;

SendDlgItemMessage(hDlg, IDC\_EDIT4 + i, WM\_SETTEXT, (WPARAM)0, (LPARAM)szVal);

}

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL) {

EndDialog(hDlg, LOWORD(wParam));

return TRUE;

}

break;

}

return FALSE;

}

// Главная функция окна диалога GetName

LRESULT CALLBACK GetName(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {

static char szVal[30];

switch (message) {

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK) {

SendDlgItemMessage(hDlg, IDC\_EDIT1, WM\_GETTEXT, (WPARAM)sizeof(szVal), (LPARAM)szVal);

std::ostrstream(leaders[gametype].name, sizeof(leaders[gametype].name)) << szVal << std::ends;

}

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL) {

EndDialog(hDlg, LOWORD(wParam));

return TRUE;

}

break;

}

return FALSE;

}

// Автомат "Управление игрой"

void ALines(int e) {

int y\_old = y\_lines;

switch (y\_lines) {

//Поиск игрового шара

case 0:

if (e == 0 && xk1()) {

z0();

y\_lines = 1;

}

break;

//Поиск куда послать игровой шар

case 1:

if (e == 0 && xk0() && x0()) {

z1\_2();

z1\_1();

y\_lines = 2;

}

else {

if (e == 0 && xk1()) {

z1\_2();

z0();

z1\_3();

}

else if (e == 0 && xk2()) {

z1\_2();

y\_lines = 0;

}

else {

if (e == 1) {

z1\_3();

}

}

}

break;

//Передвижение игрового шара

case 2:

if (e == 1 && x1() && x2()) {

z2\_2();

z2\_3();

y\_lines = 3;

}

else if (e == 1 && x1()) {

z2\_2();

z2\_4();

y\_lines = 4;

}

else if (e == 1) {

z2\_1();

}

break;

//Удаление линии

case 3:

if (e == 1 && x3()) {

z3\_2();

y\_lines = 0;

}

else if (e == 1) {

z3\_1();

}

break;

//Появление новых шаров

case 4:

if (e == 1 && x4() && x5()) {

z4\_2();

z2\_3();

y\_lines = 3;

}

else if (e == 1 && x4()) {

z4\_2();

y\_lines = 0;

}

else if (e == 1) {

z4\_1();

}

break;

}

if (y\_old != y\_lines) {

switch (y\_lines) {

case 1:

z1\_3();

break;

case 2:

z2\_1();

break;

case 3:

z3\_1();

break;

case 4:

z4\_1();

break;

}

}

}

//ВХОДНЫЕ ПЕРЕМЕННЫЕ

//Выбранная ячейка пуста

bool xk0() {

return (click\_ball.State() == 0 || click\_ball.State() == 1);

}

//В выбранной ячейке находиться шар

bool xk1() {

return (click\_ball.State() == 3);

}

//В выбранной ячейке находиться прыгающий шар

bool xk2() {

return (click\_ball.State() == 4);

}

//Существует путь от активной ячейки до выбранной

bool x0() {

return FindPath(ball, click\_ball);

}

//Передвижение закончилось

bool x1() {

return path.empty();

}

//Требуется удалить линии (после передвижения)

bool x2() {

return CheckLines(ball);

}

//Удаление закончилось

bool x3() {

itr = explode\_list.begin();

return (itr->State() == 0);//N3

}

//Появление закончилось

bool x4() {

itr = appear\_list.begin();

return (itr->State() == 3);//N1

}

//Требуется удалить линии (после появления шаров)

bool x5() {

itr = appear\_list.begin();

while (itr != appear\_list.end()) {

CheckLines(\*itr++);

}

return (explode\_list.size() != 0);

}

// ВЫХОДНЫЕ ВОЗДЕЙСТВИЯ

//Запустить прыжок

void z0()

{

SetTimer(hWnd, 1, 50, NULL);

PlaySound(MY\_SOUND0, NULL, SND\_FILENAME | SND\_ASYNC);

ball = click\_ball;

}

//Запустить передвижение

void z1\_1() {

SetTimer(hWnd, 1, 50, NULL);

ball\_color = ball.Color();

}

//Закончить прыжок

void z1\_2() {

KillTimer(hWnd, 1);

ball.ACell(6);

}

//Прыгать

void z1\_3() {

ball.ACell(5);

}

//Передвинуть шар на следующую ячейку

void z2\_1() {

ball.ACell(0);

ball = path.top();

path.pop();

ball.ACell(1);

}

//Закончить передвижение

void z2\_2() {

KillTimer(hWnd, 1);

}

//Запустить удаление

void z2\_3() {

SetTimer(hWnd, 1, 20, NULL);

PlaySound(MY\_SOUND1, NULL, SND\_FILENAME | SND\_ASYNC);

}

//Запустить появление

void z2\_4() {

CheckAppearList();

SetTimer(hWnd, 1, 50, NULL);

}

//Удалить

void z3\_1() {

itr = explode\_list.begin();

while (itr != explode\_list.end()) {

(\*itr++).ACell(4);

}

}

//Закончить удаление

void z3\_2() {

KillTimer(hWnd, 1);

gamescore += (explode\_list.size() - del\_balls + 1) \* explode\_list.size();

DrawScore();

explode\_list.clear();

}

//Появление

void z4\_1() {

PlaySound(MY\_SOUND2, NULL, SND\_FILENAME | SND\_ASYNC);

itr = appear\_list.begin();

while (itr != appear\_list.end()) {

(\*itr++).ACell(3);

}

}

//Закончить появление

void z4\_2() {

KillTimer(hWnd, 1);

GenerateAppearList();

}

//Проверка что через ячейку in проходит подлещащая удаления линия(или несколько линии)

bool CheckLines(const cell& in) {

int x = in.posx;

int y = in.posy;

int c = in.Color();

int i, j;

cell l;

bool b = false;

i = 1;

while ((x + i < max\_x)

&& (map[x + i][y].y == 3)

&& (map[x + i][y].color == c)) {

++i;

}

j = 1;

while ((x - j >= 0)

&& (map[x - j][y].y == 3)

&& (map[x - j][y].color == c)) {

++j;

}

if (j + i - 1 >= del\_balls) {

l.posx = x + i;

l.posy = y;

for (int k = 0; k < i + j - 1; k++) {

l.posx--;

explode\_list.push\_back(l);

}

b = true;

}

i = 1;

while ((y + i < max\_y)

&& (map[x][y + i].y == 3)

&& (map[x][y + i].color == c)) {

++i;

}

j = 1;

while ((y - j >= 0)

&& (map[x][y - j].y == 3)

&& (map[x][y - j].color == c)) {

++j;

}

if (j + i - 1 >= del\_balls) {

l.posx = x;

l.posy = y + i;

for (int k = 0; k < i + j - 1; k++) {

l.posy--;

explode\_list.push\_back(l);

}

b = true;

}

i = 1;

while ((x + i < max\_x)

&& (y + i < max\_y)

&& (map[x + i][y + i].y == 3)

&& (map[x + i][y + i].color == c)) {

++i;

}

j = 1;

while ((x - j >= 0)

&& (y - j >= 0)

&& (map[x - j][y - j].y == 3)

&& (map[x - j][y - j].color == c)) {

++j;

}

if (j + i - 1 >= del\_balls) {

l.posx = x + i;

l.posy = y + i;

for (int k = 0; k < i + j - 1; k++) {

l.posx--;

l.posy--;

explode\_list.push\_back(l);

}

b = true;

}

i = 1;

while ((x + i < max\_x)

&& (y - i >= 0)

&& (map[x + i][y - i].y == 3)

&& (map[x + i][y - i].color == c)) {

++i;

}

j = 1;

while ((x - j >= 0)

&& (y + j < max\_y)

&& (map[x - j][y + j].y == 3)

&& (map[x - j][y + j].color == c)) {

++j;

}

if (j + i - 1 >= del\_balls) {

l.posx = x + i;

l.posy = y - i;

for (int k = 0; k < i + j - 1; k++) {

l.posx--;

l.posy++;

explode\_list.push\_back(l);

}

b = true;

}

if (b) {

explode\_list.remove(in);

explode\_list.push\_back(in);

}

return b;

}

//Поиск пути следования шарика из ячейки from в in

bool FindPath(const cell& from, const cell& in) {

struct {

cell pred;

int mark;

}

v[MAX\_MAP\_X][MAX\_MAP\_Y];

cell k, l;

std::queue<cell> q;

for (int i = 0; i < max\_x; i++)

for (int j = 0; j < max\_y; j++)

v[i][j].mark = 0;

v[from.posx][from.posy].mark = 1;

q.push(from);

while (!q.empty()) {

k = q.front();

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

l = k;

switch (i)

{

case 0:

l.posx--;

break;

case 1:

l.posx++;

break;

case 2:

l.posy--;

break;

case 3:

l.posy++;

break;

}

if (Valid(l) && !v[l.posx][l.posy].mark) {

v[l.posx][l.posy].mark = 1;

v[l.posx][l.posy].pred = k;

q.push(l);

if (l == in) {

do {

path.push(l);

l = v[l.posx][l.posy].pred;

} while (l != from);

return true;

}

}

}

q.pop();

}

return false;

}

// Проверить список появляющихся шаров на предмент занятости соответствующей ячейки

void CheckAppearList() {

int tmp;

itr = appear\_list.begin();

while (itr != appear\_list.end()) {

if (itr->State() == 3) {

tmp = itr->PreColor();

itr->PreColor() = -1;

FindEmptyCell(\*itr);

itr->PreColor() = tmp;

(\*itr).ACell(2);

}

++itr;

}

}

//Создать список появляющихся шаров

void GenerateAppearList() {

appear\_list.clear();

cell l;

for (int i = 0; i < app\_balls; i++) {

if (FindEmptyCell(l)) {

appear\_list.push\_back(l);

l.ACell(2);

}

else return;

}

}

//Найти пустую ячейку

bool FindEmptyCell(cell& in) {

cell l;

l.posx = random(max\_x);

l.posy = random(max\_y);

if (l.State() == 0) {

in = l;

return true;

};

for (int i = 0; i < max\_x \* max\_y; i++) {

if (l.posx != max\_x - 1) {

l.posx++;

}

else if (l.posy != max\_y - 1) {

l.posy++;

l.posx = 0;

}

else {

l.posx = 0;

l.posy = 0;

};

if (l.State() == 0) {

in = l;

return true;

}

}

GameOver();

return false;

}

//Проверить можно ли через ячейку in двигатся шару

bool Valid(const cell& in)

{

return (in.posx >= 0) && (in.posx < max\_x) &&

(in.posy >= 0) && (in.posy < max\_y) &&

(in.State() == 0 || in.State() == 1);

}

//Подготовка ресурсов для новой игры

void NewGame()

{

for (int i = 0; i < max\_x; i++) {

for (int j = 0; j < max\_y; j++) {

map[i][j].y = 0;

map[i][j].color = 0;

map[i][j].pre\_color = -1;

map[i][j].num\_pic = 0;

}

}

y\_lines = 0;

gamescore = 0;

gametime = 0;

SetTimer(hWnd, 0, 1000, NULL);

cell l;

for (int i = 0; i < del\_balls; i++) {

FindEmptyCell(l);

ball\_color = random(7);

l.ACell(1);

}

GenerateAppearList();

}

//Обработка окончания игрового процесса

void GameOver() {

KillTimer(hWnd, 0);

PlaySound(MY\_SOUND3, NULL, SND\_FILENAME | SND\_ASYNC);

if (gametype < 3) {

if (leaders[gametype].score < gamescore

|| (leaders[gametype].score == gamescore && leaders[gametype].time > gametime)) {

DialogBox(hInst, (LPCTSTR)IDD\_GETNAMEBOX, hWnd, (DLGPROC)GetName);

leaders[gametype].score = gamescore;

leaders[gametype].time = gametime;

}

else {

DialogBox(hInst, (LPCTSTR)IDD\_BESTRESULTSBOX, hWnd, (DLGPROC)BestResults);

}

}

else {

MessageBox(hWnd, "Your custom game is over...", "Condolences", MB\_OK);

}

NewGame();

InvalidateRect(hWnd, NULL, FALSE);

}

//Отрисовать на табло продолжительность игры

void DrawTime()

{

SelectObject(hCompatibleDC, bmp\_numbers);

int h = gametime;

int s2 = h % 60; h /= 60;

int s1 = s2 % 10; s2 /= 10;

int m2 = h % 60; h /= 60;

int m1 = m2 % 10; m2 /= 10;

BitBlt(hDC, max\_x \* CELL\_SIZE - 29, 5, 20, 37, hCompatibleDC, s1 \* 19, 0, SRCCOPY);

BitBlt(hDC, max\_x \* CELL\_SIZE - 50, 5, 20, 37, hCompatibleDC, s2 \* 19, 0, SRCCOPY);

BitBlt(hDC, max\_x \* CELL\_SIZE - 77, 5, 20, 37, hCompatibleDC, m1 \* 19, 0, SRCCOPY);

BitBlt(hDC, max\_x \* CELL\_SIZE - 98, 5, 20, 37, hCompatibleDC, m2 \* 19, 0, SRCCOPY);

BitBlt(hDC, max\_x \* CELL\_SIZE - 125, 5, 20, 37, hCompatibleDC, h \* 19, 0, SRCCOPY);

SelectObject(hCompatibleDC, bmp\_points);

BitBlt(hDC, max\_x \* CELL\_SIZE - 56, 5, 5, 37, hCompatibleDC, 0, 0, SRCCOPY);

BitBlt(hDC, max\_x \* CELL\_SIZE - 104, 5, 5, 37, hCompatibleDC, 0, 0, SRCCOPY);

}

//Отрисовать на табло текущие очки

void DrawScore()

{

SelectObject(hCompatibleDC, bmp\_numbers);

int t = gamescore;

for (int i = 0; i < 5; i++) {

BitBlt(hDC, 100 - 21 \* i, 5, 20, 37, hCompatibleDC, (t % 10) \* 19, 0, SRCCOPY);

t /= 10;

}

}

//Отрисовать табло

void DrawTop()

{

PatBlt(hDC, 0, 0, CELL\_SIZE \* max\_x, TOP\_HEIGHT, BLACKNESS);

DrawScore();

DrawTime();

}

//Проверить заданные параметры игры

void CheckCustomParameters()

{

if (max\_x < 6) max\_x = 6;

if (max\_x > 20) max\_x = 20;

if (max\_y < 2) max\_y = 2;

if (max\_y > 12) max\_y = 12;

if (del\_balls < 2) del\_balls = 2;

if (del\_balls > (max\_x > max\_y ? max\_x : max\_y)) {

del\_balls = (max\_x > max\_y ? max\_x : max\_y);

}

if (app\_balls + del\_balls > max\_x \* max\_y)

{

app\_balls = max\_x \* max\_y - del\_balls;

}

}

//Получение информации о лучших игроках

void GetInfo()

{

FILE\* in;

if (in = fopen("leaders.dat", "rb")) {

fread(&gametype, sizeof(int), 1, in);

fread(&max\_x, sizeof(int), 1, in);

fread(&max\_y, sizeof(int), 1, in);

fread(&app\_balls, sizeof(int), 1, in);

fread(&del\_balls, sizeof(int), 1, in);

for (int i = 0; i < 3; i++) {

fread(&leaders[i], sizeof(info), 1, in);

}

fclose(in);

}

else {

gametype = 0;

max\_x = 9; max\_y = 9;

app\_balls = 3; del\_balls = 5;

for (int i = 0; i < 3; i++) {

leaders[i] = null\_leader;

}

}

}

//Сохранить информацию о лучших игроках

void WriteInfo()

{

FILE\* out;

if (out = fopen("leaders.dat", "wb")) {

fwrite(&gametype, sizeof(int), 1, out);

fwrite(&max\_x, sizeof(int), 1, out);

fwrite(&max\_y, sizeof(int), 1, out);

fwrite(&app\_balls, sizeof(int), 1, out);

fwrite(&del\_balls, sizeof(int), 1, out);

for (int i = 0; i < 3; i++) {

fwrite(&leaders[i], sizeof(info), 1, out);

}

fclose(out);

}

}