Босько Виктор, 151004, Лабораторная работа №1

**Source.cpp**

#include <windows.h>

#include <gdiplus.h>

#include "Constants.h"

#include "Variables.h"

#include "Functions.h"

#include "resource.h"

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

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

WNDCLASSEX wcex;

HWND hWnd;

MSG msg;

HACCEL hAccel;

Gdiplus::GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_DBLCLKS;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

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

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

wcex.hbrBackground = HBRUSH(CreateSolidBrush(RGB(0, 0, 0)));

wcex.lpszMenuName = NULL;

wcex.lpszClassName = ProjConst::PROJ\_NAME;

wcex.hIconSm = wcex.hIcon;

RegisterClassEx(&wcex);

hWnd = CreateWindow(ProjConst::PROJ\_NAME, ProjConst::WND\_CAPTION, WS\_OVERLAPPEDWINDOW,

100, 50, ProjConst::WND\_DEF\_WIDTH, ProjConst::WND\_DEF\_HEIGHT, NULL, NULL, hInstance, NULL);

hAccel = LoadAccelerators(hInstance, (LPCTSTR)IDR\_ACCELERATOR1);

if (hAccel == NULL) {

MessageBox(hWnd, L"Table of acceleration is null", L"Error Message", NULL);

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

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

{

if (!TranslateAccelerator(hWnd, hAccel, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

Gdiplus::GdiplusShutdown(gdiplusToken);

return (int)msg.wParam;

}

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

HDC hdc, memDC;

PAINTSTRUCT ps;

RECT rect = {ProjConst::PIC\_INITIAL\_X, ProjConst::PIC\_INITIAL\_Y,

ProjVars::x + ProjConst::PIC\_WIDTH, ProjVars::y + ProjConst::PIC\_HEIGHT};

HKL hkl;

HBITMAP memBMP;

HGDIOBJ hOld;

switch (message) {

case WM\_CHAR:

if (GetAsyncKeyState(VK\_RSHIFT)) {

ProjVars::realSpeed = ProjConst::SPEED \* 4;

}

else {

ProjVars::realSpeed = ProjConst::SPEED;

}

hkl = GetKeyboardLayout(NULL);

if (ProjFuncs::isRusLayout(hkl)) {

DrawFuncs::ProcessRusLayout(wParam);

}

else {

DrawFuncs::ProcessEngLayout(wParam);

}

rect = ProjFuncs::GetRect();

InvalidateRect(hWnd, &rect, true);

break;

case WM\_MOUSEWHEEL:

if (GetAsyncKeyState(VK\_RSHIFT)) {

ProjVars::realSpeed = ProjConst::SPEED \* 4;

}

else {

ProjVars::realSpeed = ProjConst::SPEED;

}

if (GetAsyncKeyState(VK\_LSHIFT)) {

DrawFuncs::ProcessHorisontalMouseScroll(wParam);

}

else {

DrawFuncs::ProcessVerticalMouseScroll(wParam);

}

rect = ProjFuncs::GetRect();

InvalidateRect(hWnd, &rect, true);

break;

case WM\_LBUTTONDOWN:

ProjVars::x = ProjFuncs::GetXParam(lParam);

if (ProjVars::x < ProjConst::BORDER\_LEFT) {

ProjVars::x = ProjConst::BORDER\_LEFT;

}

if (ProjVars::x > ProjConst::BORDER\_RIGHT) {

ProjVars::x = ProjConst::BORDER\_RIGHT;

}

ProjVars::y = ProjFuncs::GetYParam(lParam);

if (ProjVars::y > ProjConst::BORDER\_BOTTOM) {

ProjVars::y = ProjConst::BORDER\_BOTTOM;

}

if (ProjVars::y < ProjConst::BORDER\_TOP) {

ProjVars::y = ProjConst::BORDER\_TOP;

}

rect = {0, 0, ProjConst::WND\_DEF\_WIDTH,

ProjConst::WND\_DEF\_HEIGHT};

InvalidateRect(hWnd, &rect, true);

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case NEXT\_SPRITE:

ProjVars::currPic++;

rect = ProjFuncs::GetRect();

InvalidateRect(hWnd, &rect, true);

break;

case PREV\_SPRITE:

ProjVars::currPic--;

rect = ProjFuncs::GetRect();

InvalidateRect(hWnd, &rect, true);

break;

default:

break;

}

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

DrawFuncs::DoubleBufferedPaint(hWnd, hdc);

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

**Functions.h**

#pragma once

namespace ProjFuncs {

wchar\_t\* GetCurrPicName() {

ProjVars::currPic %= ProjConst::TOTAL\_PICTURES;

if (ProjVars::currPic < 0) {

ProjVars::currPic +=

ProjConst::TOTAL\_PICTURES;

}

return (wchar\_t\*)ProjConst::PIC\_NAMES[ProjVars::currPic];

}

bool isRusLayout(HKL hkl) {

return (int)hkl == ProjConst::RUS\_LAYOUT;

}

RECT GetRect() {

return { ProjVars::x – ProjVars::realSpeed- 1, ProjVars::y - ProjVars::realSpeed - 1,

ProjVars::x + ProjConst::PIC\_WIDTH + ProjVars::realSpeed + 1,

ProjVars::y + ProjConst::PIC\_HEIGHT + ProjVars::realSpeed + 1 };

}

int GetXParam(LPARAM lParam) {

int res = lParam & 0xFFFF;

res -= ProjConst::PIC\_WIDTH / 2;

return res;

}

int GetYParam(LPARAM lParam) {

int res = lParam >> 16;

res -= ProjConst::PIC\_HEIGHT / 2;

return res;

}

}

namespace DrawFuncs {

void DrawImage(HDC hdc, int x, int y) {

Gdiplus::Graphics graphics(hdc);

Gdiplus::Bitmap bmp(ProjFuncs::GetCurrPicName());

graphics.DrawImage(&bmp, x, y, ProjConst::PIC\_WIDTH, ProjConst::PIC\_HEIGHT);

}

void DoubleBufferedPaint(HWND hWnd, HDC hdc) {

HDC memDC;

HBITMAP memBMP;

HGDIOBJ hOld;

memDC = CreateCompatibleDC(hdc);

memBMP = CreateCompatibleBitmap(hdc, ProjConst::WND\_DEF\_WIDTH, ProjConst::WND\_DEF\_HEIGHT);

hOld = SelectObject(memDC, memBMP);

DrawImage(memDC, ProjVars::x, ProjVars::y);

BitBlt(hdc, 0, 0, ProjConst::WND\_DEF\_WIDTH, ProjConst::WND\_DEF\_HEIGHT, memDC, 0, 0, SRCCOPY);

SelectObject(memDC, hOld);

DeleteObject(memBMP);

DeleteDC(memDC);

ReleaseDC(hWnd, hdc);

}

void ProcessRusLayout(WPARAM wParam) {

//W

if (wParam == ProjConst::VK\_CAP\_RUS\_W || wParam == ProjConst::VK\_RUS\_W) {

//

if (ProjVars::y < ProjConst::BORDER\_BOTTOM) {

ProjVars::y += ProjVars::realSpeed;

}

}

//D

else if (wParam == ProjConst::VK\_CAP\_RUS\_D || wParam == ProjConst::VK\_RUS\_D) {

//

if (ProjVars::x > ProjConst::BORDER\_LEFT) {

ProjVars::x -= ProjVars::realSpeed;

}

}

//A

else if (wParam == ProjConst::VK\_CAP\_RUS\_A || wParam == ProjConst::VK\_RUS\_A) {

//

if (ProjVars::x < ProjConst::BORDER\_RIGHT) {

ProjVars::x += ProjVars::realSpeed;

}

}

//S

else if (wParam == ProjConst::VK\_CAP\_RUS\_S || wParam == ProjConst::VK\_RUS\_S) {

//

if (ProjVars::y > ProjConst::BORDER\_TOP) {

ProjVars::y -= ProjVars::realSpeed;

}

}

}

void ProcessEngLayout(WPARAM wParam) {

switch (wParam) {

case 'w':

case 'W':

if (ProjVars::y > ProjConst::BORDER\_TOP) {

ProjVars::y -= ProjVars::realSpeed;

}

break;

case 's':

case 'S':

if (ProjVars::y < ProjConst::BORDER\_BOTTOM) {

ProjVars::y += ProjVars::realSpeed;

}

break;

case 'a':

case 'A':

if (ProjVars::x > ProjConst::BORDER\_LEFT) {

ProjVars::x -= ProjVars::realSpeed;

}

break;

case 'd':

case 'D':

if (ProjVars::x < ProjConst::BORDER\_RIGHT) {

ProjVars::x += ProjVars::realSpeed;

}

break;

default:

break;

}

}

void ProcessHorisontalMouseScroll(WPARAM wParam) {

if (GET\_WHEEL\_DELTA\_WPARAM(wParam) > 0) {

if (ProjVars::x > ProjConst::BORDER\_LEFT) {

ProjVars::x -= ProjVars::realSpeed;

}

}

else {

if (ProjVars::x < ProjConst::BORDER\_RIGHT) {

ProjVars::x += ProjVars::realSpeed;

}

}

}

void ProcessVerticalMouseScroll(WPARAM wParam) {

if (GET\_WHEEL\_DELTA\_WPARAM(wParam) > 0) {

if (ProjVars::y > ProjConst::BORDER\_TOP) {

ProjVars::y -= ProjVars::realSpeed;

}

}

else {

if (ProjVars::y < ProjConst::BORDER\_BOTTOM) {

ProjVars::y += ProjVars::realSpeed;

}

}

}

}