Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и

системное программирование

ОТЧЁТ

к лабораторной работе №1

на тему

Основы программирования в Win32 API. Оконное приложение Win32 с минимальной функциональной достаточностью. Обработка основных оконных сообщений

Выполнил: студент группы 153504

Тиханёнок Илья Александрович

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146728177)

[2 Краткие теоретические сведения 4](#_Toc146728178)

[2 Результаты выполнения лабораторной работы 6](#_Toc146728179)

[Выводы 8](#_Toc146728180)

[Список использованных источников 9](#_Toc146728181)

[Приложение А (обязательное) Листинг кода 10](#_Toc146728182)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на Win32 API, использования виджетов и обработки оконных сообщений (как базовых, так и пользовательских). Реализовать вышеупомянутые требования на примере змейки, которое обрабатывает основные оконные сообщения через функцию WndProc. Основные сообщения, такие как WM\_PAINT, WM\_KEYDOWN, WM\_TIMER, WM\_COMMAND, и WM\_CLOSE, обрабатываются для управления отрисовкой игры, обновления состояния игры и обработки событий, таких как нажатие клавиш и закрытие окна.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерфейс прикладного программирования Win32 API представляет собой набор функций и подпрограмм, предоставляющих программный доступ к возможностям операционной системы Windows. Этот API предоставляет средства для управления и взаимодействия с операционной системой, а также для создания приложений, которые работают в среде Windows.

Win32 API включает в себя более 3000 функций, которые позволяют реализовать различные сервисы операционной системы. Эти функции могут быть вызваны из приложений для выполнения разнообразных задач, таких как создание процессов (например, CreateProcess) и обработка оконных сообщений (например, GetMessage).

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

­– Win32 API (Application Programming Interface): Win32 API — это набор функций и подпрограмм, предоставляемых операционной системой Windows для взаимодействия с приложениями. Он предоставляет доступ к различным функциональным возможностям Windows, таким как создание окон, обработка сообщений, работа с файлами и др. В данной лабораторной работе Win32 API использован для создания графического интерфейса калькулятора и обработки событий.

– Обработка событий и сообщений: в Windows-приложениях взаимодействие с пользователем осуществляется через обработку событий и сообщений. В коде используются функции обработки сообщений, такие как SoftwareMainProcedure, которая отвечает за обработку событий, происходящих в приложении, таких как нажатия кнопок и меню.

– Создание пользовательского интерфейса: для создания графического интерфейса приложения были использованы стандартные элементы управления Windows, такие как окна, кнопки и текстовые поля. Эти элементы были созданы и настроены с использованием Win32 API функций.

– Математические операции: для выполнения математических операций, таких как сложение, вычитание, умножение и деление, а также расчет синуса, косинуса, тангенса и котангенса, использовались стандартные функции библиотеки cmath (например, sin, cos, tan) и операции с плавающей точкой для сохранения и обработки чисел.

– Работа с окнами и элементами управления: для создания оконного приложения была использована функция CreateWindow, которая создает окно с указанными характеристиками, такими как размер, положение и стиль.

Также использовались функции для работы с элементами управления, такие как кнопки и текстовые поля.

– Обработка меню: для создания меню и обработки команд меню использовалась функция CreateMenu и связанные с ней функции для добавления пунктов меню и обработки выбора команд.

– Обработка ошибок: для обработки ошибок, таких как деление на ноль или некорректный ввод, использовались функции вывода сообщений с помощью MessageBoxA.

– Интерфейс пользователя: графический интерфейс приложения разработан с учетом стандартных принципов пользовательского интерфейса, таких как расположение элементов и размещение текста.

## 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы была разработана классическая игра "Змейка". Игра предоставляет пользователю возможность управлять змеей на игровом поле, собирать еду и увеличивать длину змеи при помощи клавиш: влево, вправо, вверх, вниз (Рисунок 1).

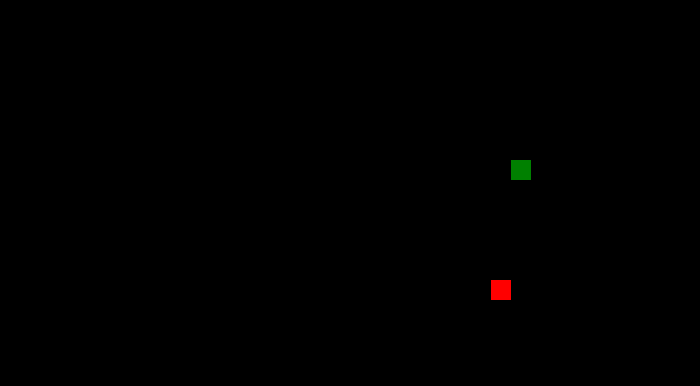


Рисунок 1 - Главное окно

Пользователь может видеть статистику собранных яблок и название нашей игры (Рисунок 2).

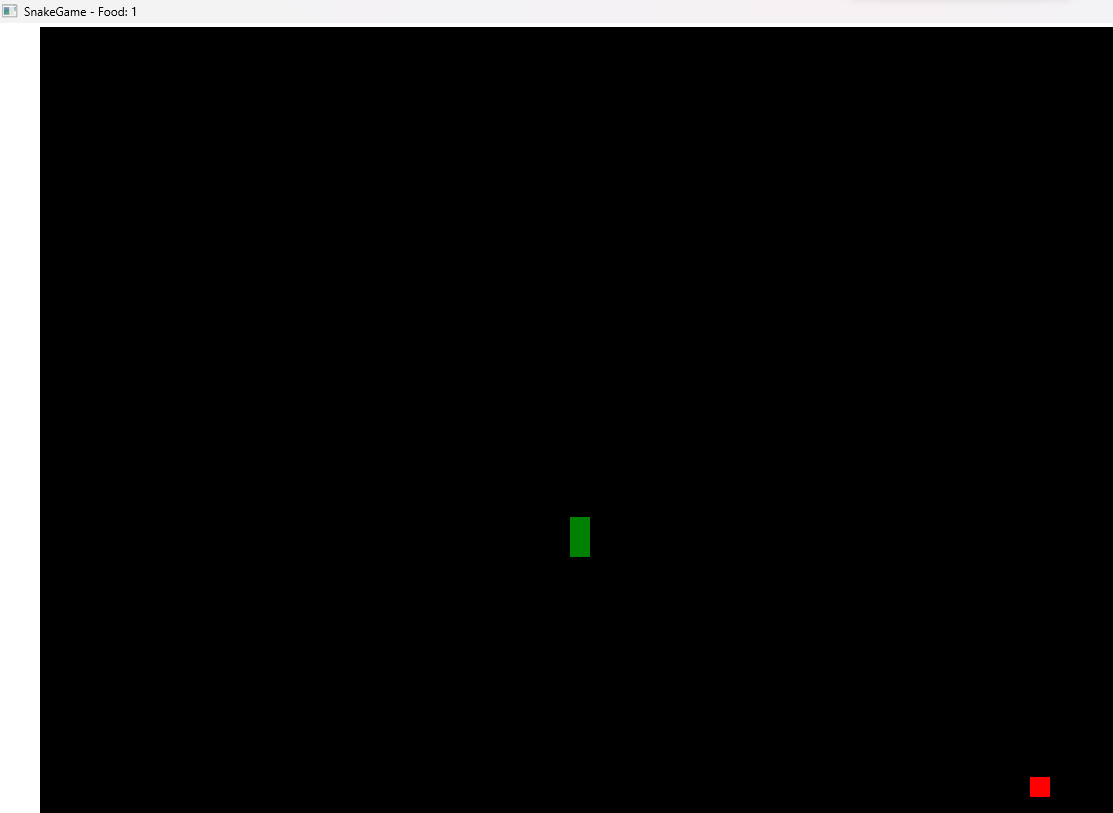


Рисунок 2 ─ Меню

Также есть система достижений, которая выскакивает каждый раз при собирании 5 яблок (Рисунок 3).



Рисунок 3 – Система достижений

Также есть система окончания игры, которая сделана в виде выпадающего окна, после того как ты врезался в край зоны игрового поля (Рисунок 4).

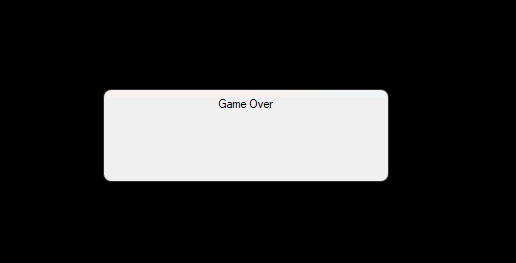


Рисунок 4 – Окно окончания игры

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы была разработана игра "Змейка" с использованием Win32 API. Проект включает в себя создание графического окна приложения, обработку клавиатурных событий для управления змейкой, отображение графики для змейки и яблок, а также внедрение некоторых дополнительных функциональных элементов. Результатом стало работоспособное простое приложение, в виде игры, написанное на языке C++, имеющее лёгкое управление даже для начинающего пользователя и способное скоротать время.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ип.

[2] Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api ─ Дата доступа 17.09.2023

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**Lab1.cpp**

#include <windows.h>

#include <vector>

#include <ctime>

#include <string>

//Эти строки подключают заголовочные файлы, необходимые для работы с Windows API и стандартной библиотекой C++.

#define MAX\_LOADSTRING 100

#define IDS\_APP\_TITLE 101

//Здесь определены макросы, которые используются позже в коде для задания максимальной длины строки и идентификатора для заголовка приложения.

HINSTANCE hInst;

HWND hWnd;

HWND hRestartButton; // Добавлено окно кнопки рестарта

//Эти переменные хранят информацию о текущем экземпляре приложения(hInst), о главном окне приложения(hWnd) и о кнопке рестарта(hRestartButton).

const int gridSize = 20;

int width = 20; // Ширина и высота поля

int height = 15;

//Здесь определены константы для размера сетки и размера поля игры в клетках.

std::vector<POINT> snake;

POINT food;

//Эти переменные хранят информацию о положении змейки (snake) и еде (food) на поле игры.

int directionX = 1;

int directionY = 0;

//Эти переменные определяют направление движения змейки по осям X и Y.Например, (1, 0) означает движение вправо, (-1, 0) - влево, (0, 1) - вниз, и(0, -1) - вверх.

bool gameOver = false;

int foodCount = 0;

//Переменные gameOver и foodCount используются для отслеживания состояния игры: завершена ли она и сколько еды съела змейка.

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

void UpdateGame();

void DrawGame(HDC hdc);

void CreateFood();

void RestartGame(); // Добавлена функция перезапуска игры

//Это прототипы функций, которые будут определены позже в коде. Они включают в себя регистрацию класса окна, инициализацию экземпляра приложения, обработчик оконных сообщений (WndProc), обновление игры, отрисовку игры, создание еды и функцию перезапуска игры.

HHOOK g\_hKeyboardHook = NULL;

//Эта переменная будет использоваться для хранения информации о глобальном хуке клавиш.

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam);

//Это прототип функции, которая будет использоваться как обработчик глобального хука клавиш.

// Глобальная переменная для хранения хендла окна сообщения

HWND g\_hMessageBox = NULL;

//Эта переменная будет хранить хендл окна сообщения, которое будет отображаться в игре.

void ShowNotification(LPCWSTR message) {

// Получить размер экрана

int screenWidth = GetSystemMetrics(SM\_CXSCREEN);

int screenHeight = GetSystemMetrics(SM\_CYSCREEN);

// Размер и положение окна сообщения

int notificationWidth = 300;

int notificationHeight = 100;

int notificationX = (screenWidth - notificationWidth) / 2;

int notificationY = (screenHeight - notificationHeight) / 2;

// Создание окна сообщения

g\_hMessageBox = CreateWindow(L"STATIC", message, WS\_POPUP | WS\_VISIBLE | SS\_CENTER | WS\_BORDER | MB\_TOPMOST,

notificationX, notificationY, notificationWidth, notificationHeight, hWnd, NULL, hInst, NULL);

// Установка таймера для закрытия окна через 5 секунды

SetTimer(hWnd, 2, 500, NULL);

// Центрирование текста в окне сообщения

SendMessage(g\_hMessageBox, WM\_SETFONT, (WPARAM)GetStockObject(DEFAULT\_GUI\_FONT), MAKELPARAM(TRUE, 0));

SendMessage(g\_hMessageBox, STM\_SETIMAGE, IMAGE\_ICON, (LPARAM)LoadIcon(NULL, IDI\_INFORMATION));

}

void CloseNotification() {

if (g\_hMessageBox != NULL) {

DestroyWindow(g\_hMessageBox);

g\_hMessageBox = NULL;

}

}

//Это основная функция приложения, которая инициализирует приложение, регистрирует класс окна, создает окно, и входит в цикл обработки сообщений.

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow) {

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

hInst = hInstance;

MyRegisterClass(hInstance);

if (!InitInstance(hInstance, nCmdShow)) {

return FALSE;

}

MSG msg;

UINT\_PTR timerId = SetTimer(hWnd, 1, 100, NULL);

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

KillTimer(hWnd, timerId);

return (int)msg.wParam;

}

//Эта функция регистрирует класс окна.

ATOM MyRegisterClass(HINSTANCE hInstance) {

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

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

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

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

wcex.lpszMenuName = nullptr;

wcex.lpszClassName = L"SnakeGame";

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

return RegisterClassExW(&wcex);

}

//Эта функция инициализирует экземпляр приложения и создает главное окно.

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {

hInst = hInstance;

WCHAR szTitle[MAX\_LOADSTRING];

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

hWnd = CreateWindow(L"SnakeGame", szTitle, WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd) {

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

width = (GetSystemMetrics(SM\_CXSCREEN) - 100) / gridSize; // Ширина поля зависит от размеров экрана

height = (GetSystemMetrics(SM\_CYSCREEN) - 100) / gridSize; // Высота поля зависит от размеров экрана

snake.push\_back({ width / 2, height / 2 });

CreateFood();

// Создание кнопки рестарта

hRestartButton = CreateWindow(L"BUTTON", L"Restart", WS\_CHILD | WS\_VISIBLE, 50, 10, 100, 30, hWnd, (HMENU)1, hInstance, NULL);

// Создание кнопки Help

CreateWindow(L"BUTTON", L"Help", WS\_CHILD | WS\_VISIBLE, 50, 50, 100, 30, hWnd, (HMENU)2, hInstance, NULL);

// Установка глобального хука на клавиши

g\_hKeyboardHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, GetModuleHandle(NULL), 0);

return TRUE;

}

//Это функция-обработчик оконных сообщений, которая обрабатывает события, такие как отрисовка окна, нажатия клавиш и другие.

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

switch (message) {

case WM\_PAINT: {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

DrawGame(hdc);

EndPaint(hWnd, &ps);

}

break;

case WM\_KEYDOWN: {

switch (wParam) {

case VK\_LEFT:

if (directionX == 0) {

directionX = -1;

directionY = 0;

}

break;

case VK\_UP:

if (directionY == 0) {

directionX = 0;

directionY = -1;

}

break;

case VK\_RIGHT:

if (directionX == 0) {

directionX = 1;

directionY = 0;

}

break;

case VK\_DOWN:

if (directionY == 0) {

directionX = 0;

directionY = 1;

}

break;

}

}

break;

case WM\_TIMER:

if (wParam == 2) {

CloseNotification(); // Закрыть окно уведомления

}

else if (gameOver) {

KillTimer(hWnd, 1);

}

else {

UpdateGame();

}

break;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case 1: // Обработка сообщений от кнопки рестарта

RestartGame();

break;

case 2: // Обработка сообщений от кнопки Help

MessageBox(hWnd, L"Game Rules:\n\nSnake control: Arrows:\n← Left\n↑ Up\n→ Right\n↓ Down\n\nCollect red squares (apples) to grow\n\nAvoid collisions with screen boundaries and yourself", L"Help", MB\_OK | MB\_ICONINFORMATION);

break;

}

break;

case WM\_CLOSE: // Обработка закрытия окна

KillTimer(hWnd, 1);

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

void UpdateGame() {

POINT newHead = snake.front();

newHead.x += directionX;

newHead.y += directionY;

snake.insert(snake.begin(), newHead);

if (newHead.x == food.x && newHead.y == food.y) {

foodCount++;

CreateFood();

WCHAR szTitle[MAX\_LOADSTRING];

wsprintf(szTitle, L"SnakeGame - Food: %d", foodCount);

SetWindowText(hWnd, szTitle);

if (foodCount % 5 == 0) {

ShowNotification(L"Congratulations! You have 5 apples!");

}

}

else {

snake.pop\_back();

}

if (newHead.x < 0 || newHead.x >= width || newHead.y < 0 || newHead.y >= height) {

gameOver = true;

}

for (size\_t i = 1; i < snake.size(); i++) {

if (snake[i].x == newHead.x && snake[i].y == newHead.y) {

gameOver = true;

}

}

if (gameOver) {

ShowNotification(L"Game Over");

}

InvalidateRect(hWnd, nullptr, TRUE);

}

void DrawGame(HDC hdc) {

HBRUSH greenBrush = CreateSolidBrush(RGB(0, 128, 0));

HBRUSH redBrush = CreateSolidBrush(RGB(255, 0, 0));

// HBRUSH whiteBrush = CreateSolidBrush(RGB(255, 255, 255)); // Создание кисти с белым цветом - для змеи

HBRUSH borderBrush = CreateSolidBrush(RGB(0, 0, 0)); // Цвет границы

HBRUSH backgroundBrush = CreateSolidBrush(RGB(255, 255, 255)); // Цвет заднего фона за границей

RECT rect;

GetClientRect(hWnd, &rect);

FillRect(hdc, &rect, backgroundBrush); // Заливаем задний фон цветом за границей

// Рассчитываем размеры и координаты игровой области

int gameAreaWidth = width \* gridSize;

int gameAreaHeight = height \* gridSize;

int borderSize = 10; // Толщина границы

int gameAreaLeft = (rect.right - gameAreaWidth) / 2;

int gameAreaTop = (rect.bottom - gameAreaHeight) / 2;

int gameAreaRight = gameAreaLeft + gameAreaWidth;

int gameAreaBottom = gameAreaTop + gameAreaHeight;

// Рисуем границу игровой области

RECT borderRect = { gameAreaLeft - borderSize, gameAreaTop - borderSize, gameAreaRight + borderSize, gameAreaBottom + borderSize };

FillRect(hdc, &borderRect, borderBrush);

// Рисуем вертикальные линии границы

for (int x = gameAreaLeft - borderSize; x <= gameAreaRight + borderSize; x += gridSize) {

MoveToEx(hdc, x, gameAreaTop - borderSize, NULL);

LineTo(hdc, x, gameAreaBottom + borderSize);

}

// Рисуем горизонтальные линии границы

for (int y = gameAreaTop - borderSize; y <= gameAreaBottom + borderSize; y += gridSize) {

MoveToEx(hdc, gameAreaLeft - borderSize, y, NULL);

LineTo(hdc, gameAreaRight + borderSize, y);

}

for (const auto& segment : snake) {

rect.left = gameAreaLeft + segment.x \* gridSize;

rect.top = gameAreaTop + segment.y \* gridSize;

rect.right = rect.left + gridSize;

rect.bottom = rect.top + gridSize;

FillRect(hdc, &rect, greenBrush);

}

rect.left = gameAreaLeft + food.x \* gridSize;

rect.top = gameAreaTop + food.y \* gridSize;

rect.right = rect.left + gridSize;

rect.bottom = rect.top + gridSize;

FillRect(hdc, &rect, redBrush);

/\*

for (const auto& segment : snake) {

int x = gameAreaLeft + segment.x \* gridSize + gridSize / 2; // Центр круга по X

int y = gameAreaTop + segment.y \* gridSize + gridSize / 2; // Центр круга по Y

int radius = gridSize / 2; // Радиус круга

HBRUSH brush = greenBrush; // Зеленый цвет для змеи

Ellipse(hdc, x - radius, y - radius, x + radius, y + radius);

}

int x = gameAreaLeft + food.x \* gridSize + gridSize / 2; // Центр круга по X

int y = gameAreaTop + food.y \* gridSize + gridSize / 2; // Центр круга по Y

int radius = gridSize / 2; // Радиус круга

HBRUSH brush = redBrush; // Красный цвет для яблока

Ellipse(hdc, x - radius, y - radius, x + radius, y + radius);

\*/

DeleteObject(greenBrush);

DeleteObject(redBrush);

DeleteObject(borderBrush);

DeleteObject(backgroundBrush);

// DeleteObject(whiteBrush); // Освобождение кисти

}

void CreateFood() {

srand(static\_cast<unsigned int>(time(nullptr)));

food.x = rand() % width;

food.y = rand() % height;

}

void RestartGame() {

if (gameOver) {

snake.clear();

snake.push\_back({ width / 2, height / 2 });

CreateFood();

foodCount = 0;

gameOver = false;

SetWindowText(hWnd, L"SnakeGame");

InvalidateRect(hWnd, nullptr, TRUE);

SetFocus(hWnd); // Вернуть фокус на окно игры

// Включить таймер снова

SetTimer(hWnd, 1, 100, NULL);

}

}

//Это функция-обработчик глобального хука клавиш, которая позволяет реагировать на определенные клавиши, например, для рестарта игры.

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode == HC\_ACTION) {

if (wParam == WM\_KEYDOWN) {

// Обработка нажатия клавиши (например, 'R' для рестарта игры)

KBDLLHOOKSTRUCT\* pKeyStruct = (KBDLLHOOKSTRUCT\*)lParam;

if (pKeyStruct->vkCode == 'R') {

// Вызывайте функцию рестарта игры здесь

RestartGame();

}

}

}

return CallNextHookEx(g\_hKeyboardHook, nCode, wParam, lParam);

}

**globals\_defines.h**

#pragma once

#define MAX\_LOADSTRING 100

#define IDS\_APP\_TITLE 101

//Здесь определены макросы, которые используются позже в коде для задания максимальной длины строки и идентификатора для заголовка приложения.