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

ОТЧЕТ

к лабораторной работе № 1
на тему «Основы программирования в Win 32 API. Оконное приложение Win 32 с минимальной достаточной функциональностью. Обработка основных оконных сообщений»

Выполнил:
студент гр 153504
Шишков В.В.

Проверил:
Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Задачей этой лабораторной работы является создание игры "Сапер" с графическим интерфейсом, который позволит игроку открывать ячейки на поле и отмечать мины. Для выполнения этого проекта мы должны освоить использование Win 32 API, разрабатывать оконные приложения и обрабатывать основные оконные сообщения.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ПО WIN32

Win32 API (Application Programming Interface) - это набор функций и интерфейсов, предоставляемых операционной системой Microsoft Windows для разработки приложений под эту операционную систему. Этот API обеспечивает программистам доступ к различным функциям и ресурсам операционной системы, таким как окна, файлы, устройства ввода/вывода, сетевые соединения и многое другое.

Win32 API был впервые представлен Microsoft вместе с операционной системой Windows NT и стал основой для разработки приложений под различные версии Windows, начиная с Windows 95 и Windows NT 3.1. Он предоставляет множество библиотек и заголовочных файлов, которые программисты используют для создания Windows-приложений на разных языках программирования, таких как C, C++, C#, и других.

С помощью WinAPI можно создавать различные оконные процедуры, диалоговые окна, программы и даже игры. Эта, скажем так, библиотека является базовой в освоении программирования Windows Forms, MFC, потому что эти интерфейсы являются надстройками этой библиотеки. Освоив её, Вы без труда будете создавать формы, и понимать, как это происходит.[1]

Окна (Windows) в Win32 API играют ключевую роль в создании графических пользовательских интерфейсов (GUI) для приложений, работающих под операционной системой Microsoft Windows. Они являются основными элементами для визуального представления и взаимодействия пользователя с приложением.

Предоставление интерфейса пользователя: Окна предоставляют пространство, где пользователи могут видеть содержимое приложения и взаимодействовать с ним. Они могут содержать элементы управления, такие как кнопки, текстовые поля, списки и другие элементы, которые пользователи могут использовать для выполнения действий и ввода данных.

Управление многозадачностью: Окна позволяют приложениям многозадачно работать в среде Windows. Каждое окно представляет собой независимую область, которую можно обновлять и управлять независимо от других окон. Это позволяет приложениям эффективно взаимодействовать с пользователем и обрабатывать множество задач одновременно.

Обработка событий: Окна могут обрабатывать события, такие как щелчки мышью, нажатия клавиш, перемещение мыши и другие действия пользователя. С помощью окон и процедур обработки сообщений приложения могут реагировать на эти события и выполнять соответствующие действия.

Создание пользовательских интерфейсов: С помощью окон разработчики могут создавать разнообразные пользовательские интерфейсы, включая

диалоговые окна, меню, панели инструментов и другие элементы, которые улучшают пользовательский опыт и облегчают навигацию в приложении.

WinMain - это функция, которая служит точкой входа для Windows-приложений, написанных с использованием Win32 API. Она выполняет начальную настройку приложения, создает окно и запускает цикл обработки сообщений.

hInstance — это дескриптор экземпляра или дескриптор модуля. Операционная система использует это значение для идентификации исполняемого файла или EXE-файла при загрузке в память. Некоторым функциям Windows требуется дескриптор экземпляра, например для загрузки значков или растровых изображений. [2]

Цикл обработки сообщений - это часть WinMain, которая начинается с вызова функции GetMessage. Она ожидает появление новых сообщений в очереди сообщений Windows. Как только сообщение появляется, оно извлекается и передается для предварительной обработки функции TranslateMessage, а затем для фактической обработки в оконную процедуру, связанную с конкретным окном приложения, с помощью функции DispatchMessage.

Оконные процедуры - это специальные функции, связанные с каждым окном в приложении, ответственные за обработку сообщений и событий, связанных с окном. Они обрабатывают сообщения, такие как события пользовательского ввода, системные сообщения и другие, в соответствии с логикой приложения.

Дескриптор окна (hWnd) - это указатель на структуру данных в операционной системе, которая содержит информацию о состоянии и характеристиках окна приложения. Обычно этот дескриптор возвращается функцией CreateWindow при создании окна, и он используется для взаимодействия с окном через соответствующие функции.[2]

CreateWindowW - это функция, которая позволяет создавать окна, устанавливая параметры, такие как размер, позиция и стиль, а также добавлять элементы интерфейса, такие как кнопки и текстовые поля.

3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОДУКТА

Реализация игры “Сапёр” прописана в файле window.cpp.

В коде игры присутсвуют константы “NumRows” и “NumCols” для создания поля. По стандарту константы имеют значение 10, однако могут быть изменены для изменения размеров поля. Также были реализованы некоторые функции, такие как “Алгоритм распространения нулей”, функция отрисовки поля и другие.

Для реализации графического интерфейса была создана оконная процедура WndProc, в которой должна происходить обработка всех поступающих сообщений. В обработчике WM_TIMER происходит создание таймера, который проверяет то, победил ли пользователь, каждые 100 милисекунд. В обработчике сообщения WM_PAINT происходит отрисовка игрового поля. В обработчике сообщения WM_DESTROY происходит освобождение памяти для массивов cellClicked, cellChecked и cellCount, также удаляется таймер и выводится сообщение о завершении игры. В обработчике сообщения WM_COMMAND происходит обработка нажатий на кнопки и ячейки поля. В обработчике сообщения WM_CREATE происходит создание всех кнопок.

После этого идут 2 обработчика нажатия кнопок мыши WM_LBUTTONDOWN и WM_RBUTTONDOWN. В них прописаны действия которые произойдут, если нажать левую или правую кнопки мыши.

Затем идет функция InitInstance которая отвечает за создание и инициализацию окна. В ней инициализируются массивы для отслеживания ячеек, генерируются мины на поле а также задается размер поля и кнопок.

В самом конце идет точка входа в приложение. **int APIENTRY _tWinMain** - Это точка входа в приложение. Эта функция вызывается при запуске приложения. Она принимает несколько параметров, включая дескриптор текущего экземпляра приложения (**hInstance**), предыдущий экземпляр приложения (**hPrevInstance**), командную строку (**lpCmdLine**), и флаги отображения окна (**nCmdShow**). **WNDCLASSEX wcex;** - Эта структура используется для описания класса окна, который зарегистрируется для создания главного окна приложения. Она содержит множество полей, таких как стиль окна (**style**), функция обработки сообщений (**lpfnWndProc**), указатель на меню (**lpszMenuName**), и так далее. Здесь она инициализируется для регистрации класса окна. **RegisterClassEx(&wcex);** - Эта функция регистрирует класс окна, описанный в структуре **wcex**, чтобы Windows знала, как создавать окна этого класса.

4 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

В данном приложении имеется игровое поле с заданными размерами, две кнопки а также счетчики открытых белых клеток и оставшихся флагов. Имеется возможность ставить флаги и открывать ячейки (рисунок 4.1).

Рисунок 4.1 – Главное окно приложения

При проигрыше или выигрыше выводится соответственное текстовое сообщение (рисунок 4.2, рисунок 4.3).

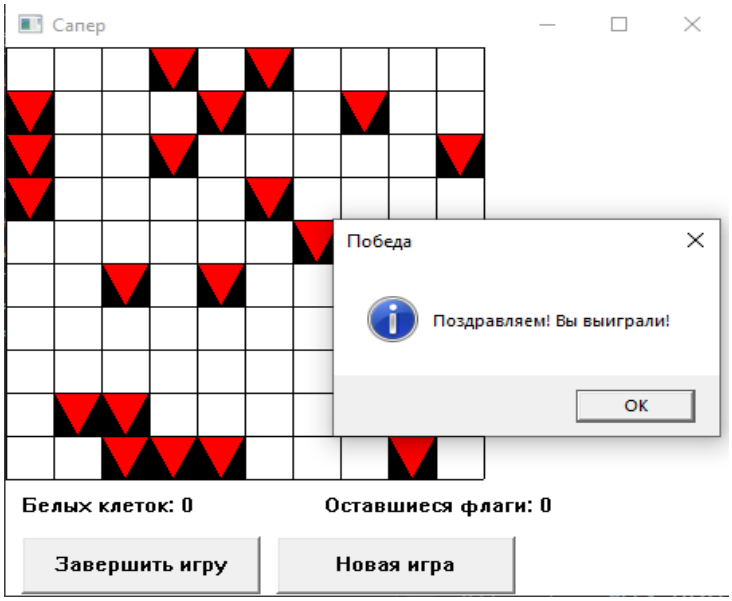

Рисунок 4.2 – Сообщение с победой

Рисунок 4.3 – Сообщение с поражением

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы был создан простой графический анимационный пример с использованием Win32 API и GDI. Эта работа позволила ознакомиться с основными концепциями программирования под Windows, включая создание окна, обработку сообщений, работу с графической библиотекой GDI, а также использование таймеров и элементов управления.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Введение в Win32 API [Электронный ресурс]. – Режим доступа: http://cppstudio.com/post/9384/

[2] Get Started with Win32 and C++ [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/winmain--the-application-entry-point

ПРИЛОЖЕНИЕ А

```
#include <windows.h>
#include <tchar.h>
#include <ctime>
#include <sstream>
#include <stack>

#define ID_BUTTON_ENDGAME 1001
#define ID_BUTTON_NEWGAME 1002

// Глобальные переменные
HINSTANCE hInst;
HWND hWndMain;
HWND hWndButtonEndGame;
HWND hWndButtonNewGame;
int cellSize = 30; // Размер ячейки
int numRows = 10;  // Количество строк
int numCols = 10;  // Количество столбцов
bool** cellClicked = nullptr; // Массив для отслеживания, была ли ячейка нажата
bool** cellChecked = nullptr; // Массив для отслеживания, была ли ячейка проверена
int** cellCount = nullptr; // Массив для счетчиков
int totalWhiteCells = 0; // Общее количество белых клеток
int score = 0; // Счетчик белых клеток
bool gameOver = false; // Флаг, указывающий на завершение игры
int numMines = 20; // Количество мин на поле
int flagsPlaced = 0; // Количество установленных флагов
bool** flags = nullptr; // Массив для отслеживания установленных флагов
bool revealMinesAlways = false; // Переменная для определения, всегда ли отображать черные клетки
int remainingFlags = numMines; // Счетчик оставшихся флагов
UINT_PTR timerID = 1; // Идентификатор таймера
const UINT timerInterval = 100; // Интервал таймера в миллисекундах (100 миллисекунд)


// Генерирует случайные черные точки на поле
void GenerateRandomBoard() {
    srand(static_cast<unsigned int>(time(nullptr))); // Инициализируем генератор случайных чисел
текущим временем

    // Заполняем поле случайными черными точками и подсчитываем белые клетки
    int numBlackPoints = 20; // Задаем количество черных точек
    totalWhiteCells = numRows * numCols - numBlackPoints;

    // Инициализируем массивы
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            cellClicked[i][j] = false;
            cellChecked[i][j] = false;
            cellCount[i][j] = 0;
        }
```

```
    }

    while (numBlackPoints > 0) {
        int x = rand() % numCols;
        int y = rand() % numRows;
        if (!cellClicked[y][x]) {
            cellClicked[y][x] = true;
            numBlackPoints--;
        }
    }

    // Вычисляем счетчики для белых клеток на основе мин в радиусе 1 от клетки
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            if (!cellClicked[i][j]) {
                // Проверяем восемь соседних клеток
                for (int dx = -1; dx <= 1; dx++) {
                    for (int dy = -1; dy <= 1; dy++) {
                        int nx = j + dx;
                        int ny = i + dy;
                        if (nx >= 0 && nx < numCols && ny >= 0 && ny < numRows && cellClicked[ny][nx]) {
                            cellCount[i][j]++;
                        }
                    }
                }
            }
        }
    }

    // Инициализируем массив для отслеживания установленных флагов
    flags = new bool* [numRows];
    for (int i = 0; i < numRows; i++) {
        flags[i] = new bool[numCols]();
    }
}

// Функция отрисовки игрового поля
void DrawBoard(HDC hdc) {
    for (int i = 0; i <= numRows; i++) {
        int y = i * cellSize;
        MoveToEx(hdc, 0, y, NULL);
        LineTo(hdc, numCols * cellSize, y);
    }
    for (int i = 0; i <= numCols; i++) {
        int x = i * cellSize;
        MoveToEx(hdc, x, 0, NULL);
        LineTo(hdc, x, numRows * cellSize);
    }


    for (int i = 0; i < numRows; i++) {
```

```
        for (int j = 0; j < numCols; j++) {
            RECT cellRect = { j * cellSize, i * cellSize, (j + 1) * cellSize, (i + 1) * cellSize };

            if (gameOver || revealMinesAlways) {
                if (cellClicked[i][j]) {
                    HBRUSH hBlackBrush = CreateSolidBrush(RGB(0, 0, 0));
                    FillRect(hdc, &cellRect, hBlackBrush);
                    DeleteObject(hBlackBrush);
                }
            }

            if (!cellClicked[i][j] && cellChecked[i][j]) {
                if (cellCount[i][j] > 0) {
                    std::wstringstream ss;
                    ss << cellCount[i][j];
                    TextOut(hdc, cellRect.left + 4, cellRect.top + 4, ss.str().c_str(),
static_cast<int>(ss.str().length()));
                }
                else {
                    // Отрисовываем "0" для белых клеток без черных соседей
                    TextOut(hdc, cellRect.left + 4, cellRect.top + 4, L"0", 1);
                }
            }
            else if (flags[i][j]) {
                HBRUSH hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
                HBRUSH hOldBrush = static_cast<HBRUSH>(SelectObject(hdc, hRedBrush));

                POINT points[3];
                points[0] = { cellRect.left, cellRect.top };
                points[1] = { cellRect.right, cellRect.top };
                points[2] = { (cellRect.left + cellRect.right) / 2, cellRect.bottom };
                Polygon(hdc, points, 3);

                SelectObject(hdc, hOldBrush);
                DeleteObject(hRedBrush);
            }
        }

    }

    // Отрисовываем счетчик белых клеток
    std::wstringstream ss;
    ss << L"Белых клеток: " << score;
    TextOut(hdc, 10, numRows * cellSize + 10, ss.str().c_str(), static_cast<int>(ss.str().length()));

    // Отрисовываем счетчик оставшихся флагов
    std::wstringstream flagsCounter;
    flagsCounter << L"Оставшиеся флаги: " << remainingFlags;
    TextOut(hdc, 200, numRows * cellSize + 10, flagsCounter.str().c_str(),
static_cast<int>(flagsCounter.str().length()));
```

```
}

// Функция проверки условия победы в игре
bool CheckWin() {
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            if (cellClicked[i][j] && cellCount[i][j] != -1 && !flags[i][j]) {
                return false; // Если есть открытая не-мина без флага, пользователь не выиграл
            }
        }
    }
    return true; // Если все открытые ячейки - мины или с флагами, пользователь выиграл
}

// Функция проверки условия победы и вывод этого
void CheckForWin() {
    if (!gameOver) {
        bool allFlagsOnMines = true; // Дополнительная переменная для проверки

        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < numCols; j++) {
                if (cellClicked[i][j] && cellCount[i][j] != -1 && !flags[i][j]) {
                    return; // Если есть открытая не-мина без флага, выходим
                }
                if (!cellClicked[i][j] && !cellChecked[i][j]) {
                    allFlagsOnMines = false; // Если есть непроверенная клетка, выставляем флаг
                }
            }
        }

        if (allFlagsOnMines) { // Если все флаги находятся на минах
            MessageBox(hWndMain, _T("Поздравляем! Вы выиграли!"), _T("Победа"), MB_OK |
MB_ICONINFORMATION);
            gameOver = true;
            //Разблокировка кнопок
            EnableWindow(hWndButtonEndGame, TRUE);
            EnableWindow(hWndButtonNewGame, TRUE);
        }
    }
}

// Функция отображения всех мин на игровом поле при завершении игры
void RevealAllMines(HDC hdc) {
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            if (cellClicked[i][j]) {
                HBRUSH hBlackBrush = CreateSolidBrush(RGB(0, 0, 0));
                HBRUSH hOldBrush = static_cast<HBRUSH>(SelectObject(hdc, hBlackBrush));

                RECT cellRect = { j * cellSize, i * cellSize, (j + 1) * cellSize, (i + 1) * cellSize };
                Ellipse(hdc, cellRect.left, cellRect.top, cellRect.right, cellRect.bottom);
```

```
            SelectObject(hdc, hOldBrush);
            DeleteObject(hBlackBrush);
        }
    }
  }
}

// Функция отображения всех мин на игровом поле при завершении игры
void SpreadZeros(int x, int y) {
  std::stack<std::pair<int, int>> zeroStack;
  zeroStack.push(std::make_pair(x, y));

  while (!zeroStack.empty()) {
    std::pair<int, int> current = zeroStack.top();
    zeroStack.pop();
    int cx = current.first;
    int cy = current.second;

    // Проверяем, что счетчик текущей белой клетки равен 0
    if (cellCount[cy][cx] == 0) {
      // Отображаем "0" для текущей белой клетки
      cellChecked[cy][cx] = true;
      score++;
      // Перерисовываем окно
      InvalidateRect(hWndMain, NULL, TRUE);

      // Перебираем восемь соседних клеток
      for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
          int nx = cx + dx;
          int ny = cy + dy;
          if (nx >= 0 && nx < numCols && ny >= 0 && ny < numRows) {
            if (!cellChecked[ny][nx]) {
              // Добавляем соседнюю клетку с нулевым счетчиком в стек
              zeroStack.push(std::make_pair(nx, ny));
            }
          }
        }
      }
    }
  }
}


void ResetGame() {
  // Освобождаем память для массивов cellClicked, cellChecked и cellCount
  for (int i = 0; i < numRows; i++) {
    delete[] cellClicked[i];
    delete[] cellChecked[i];
    delete[] cellCount[i];
```

```
    }
    delete[] cellClicked;
    delete[] cellChecked;
    delete[] cellCount;

    // Инициализируем новую игру
    cellClicked = new bool* [numRows];
    cellChecked = new bool* [numRows];
    cellCount = new int* [numRows];
    for (int i = 0; i < numRows; i++) {
        cellClicked[i] = new bool[numCols]();
        cellChecked[i] = new bool[numCols]();
        cellCount[i] = new int[numCols]();
    }

    GenerateRandomBoard(); // Генерируем случайные черные точки
    score = 0; // Сбрасываем счетчик белых клеток

    // Освобождаем память для массива flags и инициализируем его заново
    for (int i = 0; i < numRows; i++) {
        delete[] flags[i];
    }
    delete[] flags;

    flags = new bool* [numRows];
    for (int i = 0; i < numRows; i++) {
        flags[i] = new bool[numCols]();
    }
    flagsPlaced = 0;
    remainingFlags = numMines;
    // Перерисовываем окно
    InvalidateRect(hWndMain, NULL, TRUE);
}



// Функция обработки сообщений окна
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message) {
    case WM_TIMER:
        // Проверяем победу каждые 100 миллисекунд
        CheckForWin();
        break;
    case WM_PAINT: {
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hWnd, &ps);

        // Отрисовываем игровое поле
        DrawBoard(hdc);
```

```
    EndPaint(hWnd, &ps);
    break;
}

case WM_DESTROY:
    // Освобождаем память для массивов cellClicked, cellChecked и cellCount
    for (int i = 0; i < numRows; i++) {
        delete[] cellClicked[i];
        delete[] cellChecked[i];
        delete[] cellCount[i];
    }
    delete[] cellClicked;
    delete[] cellChecked;
    delete[] cellCount;

    KillTimer(hWnd, timerID);
    PostQuitMessage(0);
    break;
case WM_COMMAND:
    switch (LOWORD(wParam)) {
    case ID_BUTTON_ENDGAME:
        // Обработка нажатия кнопки "Завершить игру"
        PostQuitMessage(0);
        break;
    case ID_BUTTON_NEWGAME:
        // Обработка нажатия кнопки "Новая игра"
        ResetGame();
        gameOver = false;
        EnableWindow(hWndButtonEndGame, FALSE);
        EnableWindow(hWndButtonNewGame, FALSE);
        break;
    }
    break;
case WM_CREATE:
    EnableWindow(hWndButtonEndGame, FALSE);
    EnableWindow(hWndButtonNewGame, FALSE);
    break;
case WM_LBUTTONDOWN:
    if (!gameOver) {
        int x = LOWORD(lParam) / cellSize;
        int y = HIWORD(lParam) / cellSize;

        // Проверяем, что нажатие произошло в пределах игрового поля
        if (x >= 0 && x < numCols && y >= 0 && y < numRows) {
            if (cellClicked[y][x]) {
                gameOver = true;
                EnableWindow(hWndButtonEndGame, TRUE);
                EnableWindow(hWndButtonNewGame, TRUE);

                HDC hdc = GetDC(hWnd);
                RevealAllMines(hdc);
```

```
                ReleaseDC(hWnd, hdc);

                MessageBox(hWnd, _T("Game Over"), _T("Сообщение"), MB_OK | MB_ICONERROR);
            }
            else if (!cellChecked[y][x]) {
                if (cellCount[y][x] == 0) {
                    // Распространение нулей
                    SpreadZeros(x, y);
                }
                else {
                    // Отображаем счетчик для белых клеток
                    cellChecked[y][x] = true;
                    score++;
                    // Перерисовываем окно
                    HDC hdc = GetDC(hWnd);
                    DrawBoard(hdc); // Перерисовываем поле
                    ReleaseDC(hWnd, hdc);
                }
            }
        }
    }
    break;

case WM_RBUTTONDOWN:
    if (!gameOver) {
        int x = LOWORD(lParam) / cellSize;
        int y = HIWORD(lParam) / cellSize;

        // Проверяем, что нажатие произошло в пределах игрового поля
        if (x >= 0 && x < numCols && y >= 0 && y < numRows) {
            // Если флаг уже установлен, убираем его и увеличиваем счетчик оставшихся флагов
            if (flags[y][x]) {
                flags[y][x] = false;
                flagsPlaced--;
                remainingFlags++;
            }
            // Иначе устанавливаем флаг и уменьшаем счетчик оставшихся флагов
            else if (flagsPlaced < numMines && remainingFlags > 0) {
                flags[y][x] = true;
                flagsPlaced++;
                remainingFlags--;
            }

            // Перерисовываем окно
            InvalidateRect(hWnd, NULL, TRUE);

            // Проверяем, выиграл ли игрок после каждого хода
            if (flagsPlaced == numMines && CheckWin()) {
                MessageBox(hWnd, _T("Поздравляем! Вы выиграли!"), _T("Победа"), MB_OK |
MB_ICONINFORMATION);
                gameOver = true; // Устанавливаем флаг "Game Over"
```

```
                EnableWindow(hWndButtonEndGame, TRUE); // Кнопка "Завершить игру" доступна
                EnableWindow(hWndButtonNewGame, TRUE); // Кнопка "Новая игра" доступна
            }
        }
    }
    break;


  default:
    return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}

// Функция создания и инициализации окна
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {
  hInst = hInstance;

  // Инициализируем массивы для отслеживания нажатых ячеек, проверенных ячеек и счетчиков
  cellClicked = new bool* [numRows];
  cellChecked = new bool* [numRows];
  cellCount = new int* [numRows];
  for (int i = 0; i < numRows; i++) {
    cellClicked[i] = new bool[numCols]();
    cellChecked[i] = new bool[numCols]();
    cellCount[i] = new int[numCols]();
  }

  GenerateRandomBoard(); // Генерируем случайные черные точки

  timerID = SetTimer(hWndMain, 1, timerInterval, NULL);
  if (timerID == 0) {
    MessageBox(hWndMain, _T("Не удалось создать таймер."), _T("Ошибка"), MB_OK |
MB_ICONERROR);
    return FALSE;
  }

  hWndMain = CreateWindow(
    _T("SapperApp"), _T("Сапер"), WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT, static_cast<int>(numCols * cellSize * 1.5) + 20, numRows
* cellSize + 120, NULL, NULL, hInstance, NULL);


  hWndButtonEndGame = CreateWindow(
    _T("BUTTON"), _T("Завершить игру"), WS_CHILD | WS_VISIBLE,
    10, numRows * cellSize + 40, 150, 40, hWndMain, (HMENU)ID_BUTTON_ENDGAME, hInstance,
NULL);

  hWndButtonNewGame = CreateWindow(
    _T("BUTTON"), _T("Новая игра"), WS_CHILD | WS_VISIBLE,
```

```
        170, numRows * cellSize + 40, 150, 40, hWndMain, (HMENU)ID_BUTTON_NEWGAME, hInstance,
NULL);


    if (!hWndMain) {
        return FALSE;
    }

    ShowWindow(hWndMain, nCmdShow);
    UpdateWindow(hWndMain);

    return TRUE;
}

// Точка входа в приложение
int APIENTRY _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int
nCmdShow) {
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // Регистрируем класс окна
    WNDCLASSEX 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(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = NULL;
    wcex.lpszClassName = _T("SapperApp");
    wcex.hIconSm = LoadIcon(wcex.hInstance, IDI_APPLICATION);

    if (!RegisterClassEx(&wcex)) {
        return FALSE;
    }

    // Инициализируем главное окно приложения
    if (!InitInstance(hInstance, nCmdShow)) {
        return FALSE;
    }

    // Основной цикл обработки сообщений
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
```

```
    return (int)msg.wParam;
}
```