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

Рогачёв А.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146720283)

[2 Теоретические сведения 4](#_Toc146720284)

[3 Реализация программного продукта 5](#_Toc146720285)

[4 Результат выполнения программы 6](#_Toc146720286)

[Список использованных источников 9](#_Toc146720287)

[Приложение А 10](#_Toc146720288)

# **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 были освоены использование Win 32 API, разработка простейших оконных приложения и обработка основных оконных сообщений.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Win32 API это набор функций и структур, предоставляемых операционной системой Windows для разработки приложений под эту платформу. Win32 API предоставляет доступ к различным системным ресурсам и функциям, включая работу с окнами и оконными сообщениями.

Важнейшей концепцией в Win32 API является окно (Window). Окно - это область экрана, в которой отображается содержимое приложения. Окна могут быть разных типов, например, главные окна приложений, диалоговые окна и т.д. Каждое окно имеет уникальный идентификатор, называемый хэндлом (Handle).

Для создания окна используется функция CreateWindowEx, которая принимает параметры, такие как класс окна, заголовок, стиль и размеры. После создания окна, оно должно обрабатывать оконные сообщения.

Оконные сообщения - это как письма от других частей приложения. Они могут сообщать о событиях, таких как нажатие клавиши, клик мыши и другие действия пользователя.

Обработка оконных сообщений выполняется в функции, называемой оконной процедурой (Window Procedure). Эта функция ассоциируется с каждым окном и вызывается для обработки сообщений, отправленных окну. Она может обрабатывать сообщения различных типов.

Например, обработка сообщений WM\_CREATE, WM\_PAINT и WM\_CLOSE. Они отвечают за создание окна, перерисовку и закрытие.

# **3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОДУКТА**

Код прописанный в Main Interface.cpp представляет собой простую реализацию игры "Сапёр" на платформе Windows с использованием библиотеки WinAPI. Он создает графический интерфейс для игры и обрабатывает пользовательский ввод для управления игровым процессом.

Начинается с подключения необходимых библиотек и определения глобальных переменных для установки размеров сетки, количества мин, размера ячейки и других игровых параметров.

Затем инициализируются векторы isMine, isFlagged, isChecked, которые будут отслеживать состояние каждой ячейки сетки.

В функции PlaceMines случайным образом расставляются мины в сетке.

В функции DrawGridCells происходит отрисовка игровой сетки, включая ячейки, их состояния (открыта, флажок, не открыта) и количество мин в соседних ячейках.

Функция CheckWinCondition проверяет, выиграна ли игра, сравнивая состояния ячеек с заданными условиями победы.

Функции YouWin и YouDied отвечают за обновление текста на экране при выигрыше и проигрыше соответственно.

Функция RevealEmptyCells рекурсивно открывает пустые ячейки и их соседние ячейки, если вокруг них нет мин.

Главная функция WinMain инициализирует оконный класс, создает окно и начинает цикл обработки сообщений Windows, который обрабатывает пользовательский ввод и обновляет игровой интерфейс.

В функции WndProc обрабатываются различные сообщения Windows, такие как создание окна, команды от кнопок, отрисовка игровой сетки и обработка действий при щелчках мыши (левой и правой).

# **4 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

В данном приложении имеется игровое поле с заданными в коде размерами, кнопка restart, отвечающая за перезапуск поля, кнопка выхода exit, а также счетчик оставшихся мин. Имеется возможность помечать ячейки и открывать ячейки (рисунок 4.1).

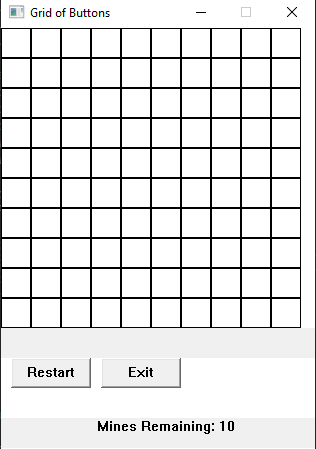


Рисунок 4.1 – Главное окно приложения на старте

При нажатии на пустую ячейку, не имеющую рядом мин, открываются все ближайшие пустые ячейки (рисунок 4.2).

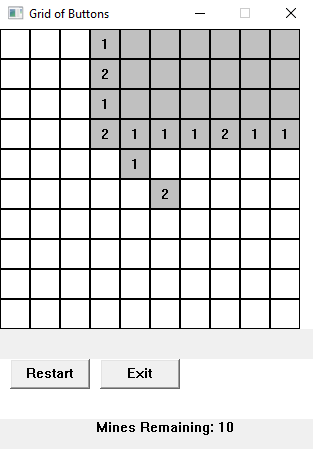


Рисунок 4.2 – Поле после открытия нескольких клеток

При проигрыше или выигрыше выводится сообщение и закрывается взаимодействие с полем (рисунок 4.3, рисунок 4.4).

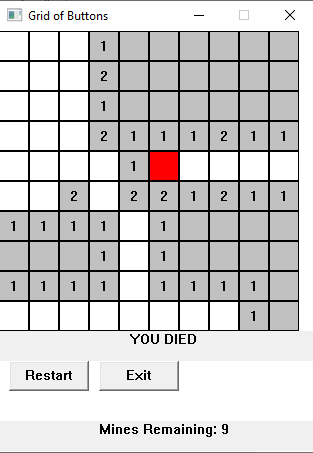


Рисунок 4.3 – Сообщение с поражением

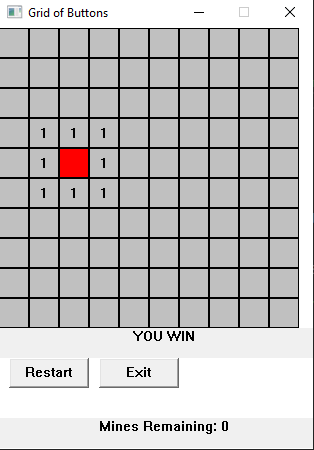


Рисунок 4.4 – Сообщение с победой

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Get Started with Win32 and C++ [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/learn-to-program-for-windows

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

[3] Создание классических приложений для Windows с помощью API Win32 – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

#include <windows.h>

#include <windowsx.h>

#include <vector>

#include <ctime>

// Define grid size and mine count

const int GRID\_SIZE = 10;

const int NUM\_MINES = 10; // You can adjust the number of mines as needed

const int CELL\_SIZE = 30; // Define cell size

int windowWidth = CELL\_SIZE \* GRID\_SIZE + 30;

int windowHeight = CELL\_SIZE \* GRID\_SIZE + 160;

bool GameOver = false;

int remainingMines = NUM\_MINES;

int flaggedMines = 0;

std::vector<bool> isMine(GRID\_SIZE\* GRID\_SIZE, false);

std::vector<bool> isFlagged(GRID\_SIZE\* GRID\_SIZE, false);

std::vector<bool> isChecked(GRID\_SIZE\* GRID\_SIZE, false);

HWND hLabel = NULL; // Global variable to store the label control handle

HWND hMineLabel = NULL;

HWND hRestartButton = NULL;

HWND hExitButton = NULL;

// Function to randomly place mines on the grid

void PlaceMines() {

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

int minesPlaced = 0;

while (minesPlaced < NUM\_MINES) {

int randomPosition = rand() % (GRID\_SIZE \* GRID\_SIZE);

if (!isMine[randomPosition]) {

isMine[randomPosition] = true;

minesPlaced++;

}

}

}

// Function to draw the grid cells

void DrawGridCells(HDC hdc) {

for (int row = 0; row < GRID\_SIZE; ++row) {

for (int col = 0; col < GRID\_SIZE; ++col) {

// Calculate cell position

int x = col \* CELL\_SIZE;

int y = row \* CELL\_SIZE;

int dr[] = { -1, -1, -1, 0, 0, 1, 1, 1 };

int dc[] = { -1, 0, 1, -1, 1, -1, 0, 1 };

int mineCount = 0;

for (int i = 0; i < 8; ++i) {

int newRow = row + dr[i];

int newCol = col + dc[i];

// Check if the neighboring cell is within bounds

if (newRow >= 0 && newRow < GRID\_SIZE && newCol >= 0 && newCol < GRID\_SIZE) {

int cellId = newRow \* GRID\_SIZE + newCol;

if (isMine[cellId]) {

mineCount++;

}

}

}

// Create a white brush for non-flagged cells

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

// Check if the cell is flagged

int cellId = row \* GRID\_SIZE + col;

if (isFlagged[cellId]) {

// Change the brush to red for flagged cells

DeleteObject(hBrush); // Delete the previous white brush

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

}

else if (isChecked[cellId])

{

DeleteObject(hBrush); // Delete the previous white brush

hBrush = CreateSolidBrush(RGB(192, 192, 192));

}

RECT cellRect = { x, y, x + CELL\_SIZE, y + CELL\_SIZE }; // Create a RECT structure

// Fill the cell with the selected brush color

FillRect(hdc, &cellRect, hBrush);

if (mineCount > 0 && isChecked[cellId]) {

// Display mine count in the cell for cells with nearby mines

SetBkMode(hdc, TRANSPARENT);

wchar\_t mineCountStr[2];

swprintf\_s(mineCountStr, L"%d", mineCount);

DrawText(hdc, mineCountStr, -1, &cellRect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);

}

// Draw an outline around the cell

FrameRect(hdc, &cellRect, CreateSolidBrush(RGB(0, 0, 0)));

// Delete the brush

DeleteObject(hBrush);

}

}

}

bool CheckWinCondition() {

for (int row = 0; row < GRID\_SIZE; ++row) {

for (int col = 0; col < GRID\_SIZE; ++col) {

int cellId = row \* GRID\_SIZE + col;

// If any empty cell is not revealed or any mine is not flagged, the game is not won

if ((!isMine[cellId] && isFlagged[cellId]) || (isMine[cellId] && !isFlagged[cellId])) {

return false;

}

}

}

return true;

}

void YouWin() {

SetWindowText(hLabel, L"YOU WIN");

GameOver = true;

}

// Function to reveal empty cells and count mines in the vicinity

void RevealEmptyCells(int row, int col, HDC hdc) {

// Array of neighboring offsets (8 positions)

int dr[] = { -1, -1, -1, 0, 0, 1, 1, 1 };

int dc[] = { -1, 0, 1, -1, 1, -1, 0, 1 };

int mineCount = 0;

for (int i = 0; i < 8; ++i) {

int newRow = row + dr[i];

int newCol = col + dc[i];

// Check if the neighboring cell is within bounds

if (newRow >= 0 && newRow < GRID\_SIZE && newCol >= 0 && newCol < GRID\_SIZE) {

int cellId = newRow \* GRID\_SIZE + newCol;

if (isMine[cellId]) {

mineCount++;

}

}

}

// Calculate cell position

int x = col \* CELL\_SIZE;

int y = row \* CELL\_SIZE;

// Create a white brush for non-flagged cells

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

RECT cellRect = { x, y, x + CELL\_SIZE, y + CELL\_SIZE };

// Gray out revealed cells

hBrush = CreateSolidBrush(RGB(192, 192, 192));

// Fill the cell with the selected brush color

FillRect(hdc, &cellRect, hBrush);

if (mineCount > 0) {

// Display mine count in the cell for cells with nearby mines

SetBkMode(hdc, TRANSPARENT);

wchar\_t mineCountStr[2];

swprintf\_s(mineCountStr, L"%d", mineCount);

DrawText(hdc, mineCountStr, -1, &cellRect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);

}

// Draw an outline around the cell

FrameRect(hdc, &cellRect, CreateSolidBrush(RGB(0, 0, 0)));

// Delete the brush

DeleteObject(hBrush);

// Recursively reveal neighboring empty cells if mine count is 0

if (mineCount == 0) {

for (int i = 0; i < 8; ++i) {

int newRow = row + dr[i];

int newCol = col + dc[i];

// Check if the neighboring cell is within bounds

if (newRow >= 0 && newRow < GRID\_SIZE && newCol >= 0 && newCol < GRID\_SIZE) {

int cellId = newRow \* GRID\_SIZE + newCol;

if (!isFlagged[cellId] && !isChecked[cellId] && !isMine[cellId]) {

isChecked[cellId] = true;

RevealEmptyCells(newRow, newCol, hdc);

}

}

}

}

}

// Function to show "YOU DIED" on the label and lock gameplay

void YouDied() {

SetWindowText(hLabel, L"YOU DIED");

GameOver = true;

}

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

switch (message) {

case WM\_CREATE:

// Create the "Restart" button

hRestartButton = CreateWindow(L"BUTTON", L"Restart", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

10, windowHeight - 130, 80, 30, hWnd, (HMENU)1001, NULL, NULL);

// Create the "Exit" button

hExitButton = CreateWindow(L"BUTTON", L"Exit", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

100, windowHeight - 130, 80, 30, hWnd, (HMENU)1002, NULL, NULL);

hLabel = CreateWindow(L"STATIC", L"", WS\_CHILD | WS\_VISIBLE | SS\_CENTER,

0, windowHeight - 160, windowWidth, 30, hWnd, NULL, NULL, NULL);

hMineLabel = CreateWindow(L"STATIC", L"", WS\_CHILD | WS\_VISIBLE | SS\_CENTER,

0, windowHeight - 70, windowWidth, 30, hWnd, NULL, NULL, NULL);

SetWindowText(hLabel, L""); // Initially empty text

wchar\_t mineCountStr[50];

swprintf\_s(mineCountStr, L"Mines Remaining: %d", NUM\_MINES - flaggedMines);

SetWindowText(hMineLabel, mineCountStr);

PlaceMines(); // Randomly place mines

break;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case 1001: // Restart button

// Reset game variables and redraw the grid

GameOver = false;

isMine.assign(GRID\_SIZE \* GRID\_SIZE, false);

isFlagged.assign(GRID\_SIZE \* GRID\_SIZE, false);

isChecked.assign(GRID\_SIZE \* GRID\_SIZE, false);

PlaceMines();

InvalidateRect(hWnd, NULL, FALSE);

SetWindowText(hLabel, L""); // Clear the label

remainingMines = NUM\_MINES;

flaggedMines = 0;

wchar\_t mineCountStr[50];

swprintf\_s(mineCountStr, L"Mines Remaining: %d", NUM\_MINES - flaggedMines);

SetWindowText(hMineLabel, mineCountStr);

break;

case 1002: // Exit button

// Close the application

PostQuitMessage(0);

break;

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

DrawGridCells(hdc); // Draw the grid cells

EndPaint(hWnd, &ps);

}

break;

case WM\_LBUTTONDOWN:

// Handle left mouse click

if (!GameOver) {

int xPos = LOWORD(lParam) / CELL\_SIZE;

int yPos = HIWORD(lParam) / CELL\_SIZE;

int cellId = yPos \* GRID\_SIZE + xPos;

if (xPos >= GRID\_SIZE || yPos >= GRID\_SIZE)

break;

if (!isFlagged[cellId] && !isChecked[cellId]) {

if (!isMine[cellId]) {

// Reveal the clicked empty cell and its neighbors

isChecked[cellId] = true;

HDC hdc = GetDC(hWnd);

RevealEmptyCells(yPos, xPos, hdc);

ReleaseDC(hWnd, hdc);

}

else {

YouDied();

}

}

if (CheckWinCondition()) {

YouWin(); // Check for win condition after each move

}

}

break;

case WM\_RBUTTONDOWN:

// Handle right mouse click

if (!GameOver) {

int xPos = LOWORD(lParam) / CELL\_SIZE;

int yPos = HIWORD(lParam) / CELL\_SIZE;

int cellId = yPos \* GRID\_SIZE + xPos;

if (!isFlagged[cellId] && !isChecked[cellId]) {

// Mark the cell as a mine

isFlagged[cellId] = true;

flaggedMines++;

wchar\_t mineCountStr[50];

swprintf\_s(mineCountStr, L"Mines Remaining: %d", NUM\_MINES - flaggedMines);

SetWindowText(hMineLabel, mineCountStr);

// Redraw the cell to indicate flagging

InvalidateRect(hWnd, NULL, FALSE);

}

else

{

isFlagged[cellId] = false;

flaggedMines--;

wchar\_t mineCountStr[50];

swprintf\_s(mineCountStr, L"Mines Remaining: %d", NUM\_MINES - flaggedMines);

SetWindowText(hMineLabel, mineCountStr);

InvalidateRect(hWnd, NULL, FALSE);

}

}

if (CheckWinCondition()) {

YouWin(); // Check for win condition after each move

}

break;

case WM\_DESTROY:

// Post a quit message to exit the application

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

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

// Define the window class

WNDCLASSEX wcex = { sizeof(WNDCLASSEX), CS\_HREDRAW | CS\_VREDRAW, WndProc, 0, 0, GetModuleHandle(NULL), NULL, NULL, NULL, NULL, L"GridWindowClass", NULL };

// Register the window class

RegisterClassEx(&wcex);

// Create the application window with a fixed size

HWND hWnd = CreateWindow(L"GridWindowClass", L"Grid of Buttons", WS\_OVERLAPPEDWINDOW & ~WS\_THICKFRAME & ~WS\_MAXIMIZEBOX,

CW\_USEDEFAULT, CW\_USEDEFAULT, windowWidth, windowHeight, NULL, NULL, hInstance, NULL);

if (!hWnd) {

return -1;

}

// Show and update the window

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

// Message loop

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}