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

Гайкевич Е.В.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[Цель работы 3](#_Toc146680819)

[Краткие теоретические сведения 4](#_Toc146680820)

[Реализация программного продукта 6](#_Toc146680821)

[Результат выполнения программы 7](#_Toc146680822)

[Список использованных источников 9](#_Toc146680823)

[Приложение А 10](#_Toc146680824)

# 1 ЦЕЛЬ РАБОТЫ

Необходимо реализовать игру "Сапер" с графическим интерфейсом, позволяющим пользователю открывать ячейки поля и помечать мины. Для разработки нужно научиться использовать Win 32 API, создавать оконные приложения и обрабатывать основные оконные сообщения.

# 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Application Programming Interface) представляет собой набор функций и подпрограмм, созданных для взаимодействия с операционной системой Windows. Этот набор программных средств обеспечивает доступ к различным функциям и сервисам операционной системы Windows, что позволяет приложениям взаимодействовать с операционной системой и использовать ее возможности.

Окна играют важнейшую роль в разработке приложений. Они являются ключевыми элементами пользовательского интерфейса и обеспечивают взаимодействие приложения с пользователем. Win32 API предоставляет обширный набор функций и возможностей для создания, управления и настройки окон. Эти функции позволяют создавать главные окна приложений, диалоговые окна, элементы управления, такие как кнопки и текстовые поля, а также обрабатывать пользовательские события, такие как клики мыши и нажатия клавиш.

Функция wWinMain является точкой входа для Windows-приложений, написанных с использованием Win32 API. Она выполняет инициализацию приложения, создание окна и запуск цикла обработки сообщений.

Дескриптор окна (hWnd) — это указатель на структуру данных внутри операционной системы, которая содержит информацию о состоянии и характеристиках окна. Этот дескриптор обычно возвращается функцией CreateWindow при создании окна. Для выполнения операций с окном разработчики передают этот дескриптор в соответствующие функции.

Для создания окон используется функция CreateWindowW. Данная функция позволяет задавать параметры окна, такие как его размер, местоположение и стиль, и создавать различные элементы интерфейса, такие как кнопки и текстовые поля.

В wWinMain происходит цикл обработки сообщений. Он начинается с вызова функции GetMessage, которая ожидает появления новых сообщений в очереди сообщений Windows. Как только сообщение появляется, оно извлекается и передается функции TranslateMessage, которая выполняет необходимую предобработку сообщения, например, преобразование клавиш в символы. Затем сообщение передается функции DispatchMessage, которая направляет его на обработку в оконную процедуру, связанную с конкретным окном приложения. Здесь происходит фактическая обработка событий и выполнение соответствующих действий, например, обновление интерфейса, обработка пользовательского ввода и многое другое.

Функции оконных процедур, также известные как оконные процедуры или оконные обработчики, представляют собой специальные функции, которые связываются с каждым окном в приложении и отвечают за обработку сообщений и событий, связанных с этим окном.

Оконные процедуры получают на вход сообщения от операционной системы и обрабатывают их в соответствии с логикой приложения. Сообщения могут быть разнообразными: от событий пользовательского ввода, таких как нажатия клавиш и перемещения мыши, до системных сообщений, связанных с жизненным циклом окон и изменением их состояния.

Оконные процедуры обычно содержат инструкции для обновления графического интерфейса приложения, обработки ввода пользователя, валидации данных, реагирования на системные команды и многое другое. Каждое окно в приложении имеет свою собственную оконную процедуру.

WM-команды (Windows Messages) представляют собой набор стандартных сообщений, используемых в Windows-приложениях для обмена информацией и управления окнами и элементами интерфейса. Эти сообщения передаются между различными компонентами приложения, включая оконные процедуры (оконные обработчики) и систему Windows, и позволяют реализовать множество функциональных возможностей.

Примеры часто используемых WM-команд:

* WM\_CREATE: Это сообщение отправляется при создании окна и предоставляет возможность для настройки и инициализации оконных данных и ресурсов.
* WM\_DESTROY: Это сообщение отправляется перед уничтожением окна. Здесь происходит освобождение ресурсов и завершение работы с окном.
* WM\_COMMAND: Сообщение WM\_COMMAND генерируется в ответ на действия пользователя, такие как нажатие кнопок и выбор элементов меню. Оно позволяет приложению реагировать на команды пользователя.
* WM\_PAINT: Это сообщение генерируется, когда окно требует перерисовки, например, после изменения размера или при необходимости отображения новой информации.

# 3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОДУКТА

Для реализации игры «Сапёр» в файлах «GamaData.h» и «GameData.cpp» были реализованы основные структуры и логика игры. В константах boardWidth, boardHeight и numberOfMines можно задавать ширину поля, высоту поля и количество минированных ячеек соответственно.

Для реализации графического интерфейса для главного окна была создана оконная процедура WndProc, в которой должна происходить обработка всех поступающих сообщений.

Для визуализации ячеек поля используются кнопки. В обработчике сообщения WM\_CREATE происходит создание всех кнопок, в том числе присвоение им уникального идентификатора и позиционирование их в главном окне.

В обработчике сообщения WM\_COMMAND происходит обработка нажатий на кнопки (ячейки). С помощью wParam находится уникальный идентификатор кнопки, на которую было произведено нажатие. Благодаря нему находится точная позиция нужной ячейки. Затем проверяется нажатие на клавишу «shift». Если во время клика по кнопке данная клавиша была зажата на клавиатуре, то вызовется функция установки флажка, иначе будет вызвана функция открытия ячейки. После каждого вызова данной функции происходит проверка на выигрыш или проигрыш. В случае, если одно из этих действий свершилось, с помощью команды MessageBox будет вызвано информационное окно с данной информацией. После нажатия на ячейку вызывается функция InvalidateRect для вызова сообщения WM\_PAINT и перерисовки кнопок.

В обработчике сообщения WM\_PAINT происходит отрисовка кнопок с заданным текстом, который зависит от того, открыта ли ячейка, помечена ли она флажком и есть ли в ней мина.

# 4 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

В выполненном приложении имеется понятная структура кнопок, напоминающая сетку. Имеется возможность ставить флаги и открывать ячейки. На кнопки устанавливаются надписи в зависимости от состояния ячеек (рисунок 4.1).

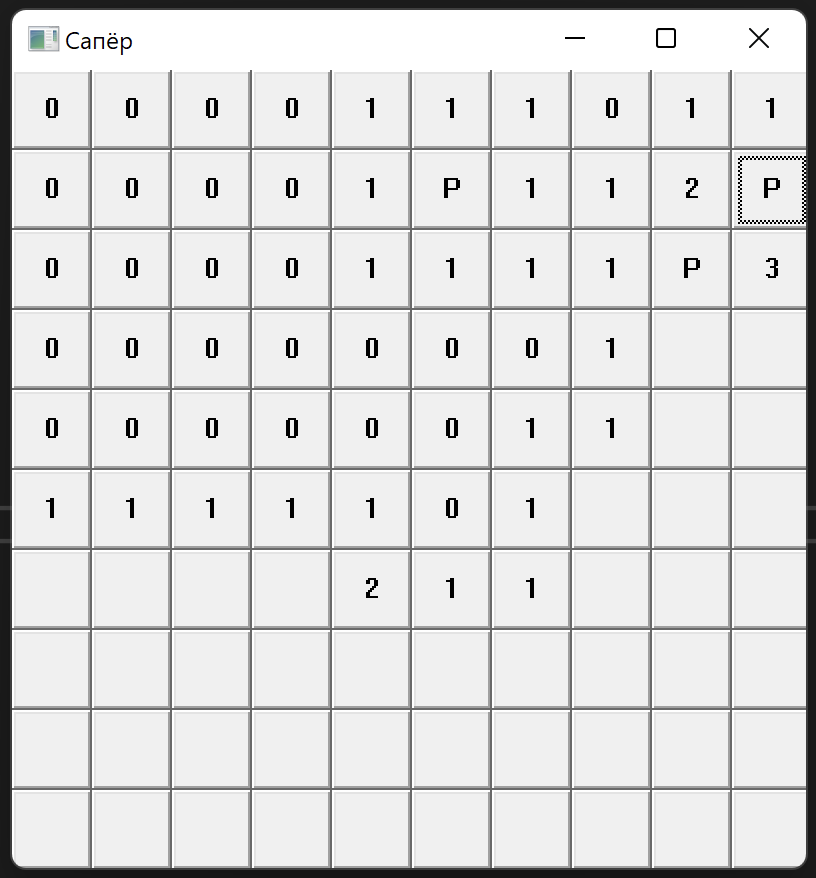


Рисунок 4.1 – Главное окно приложения

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

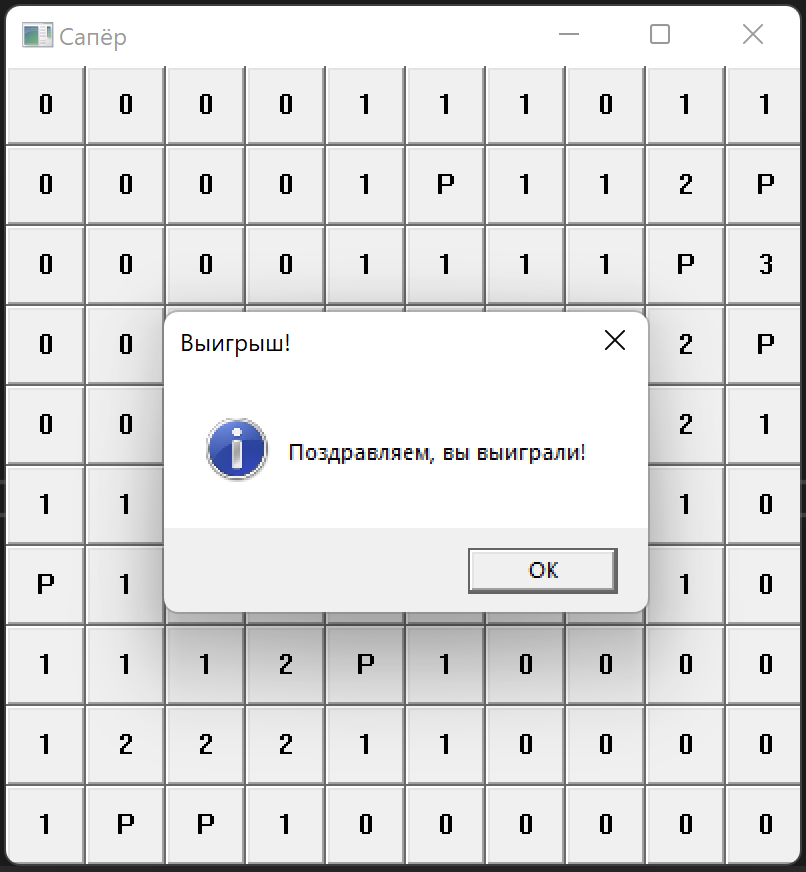


Рисунок 3.2 – Текстовое сообщение, информирующее о выигрыше

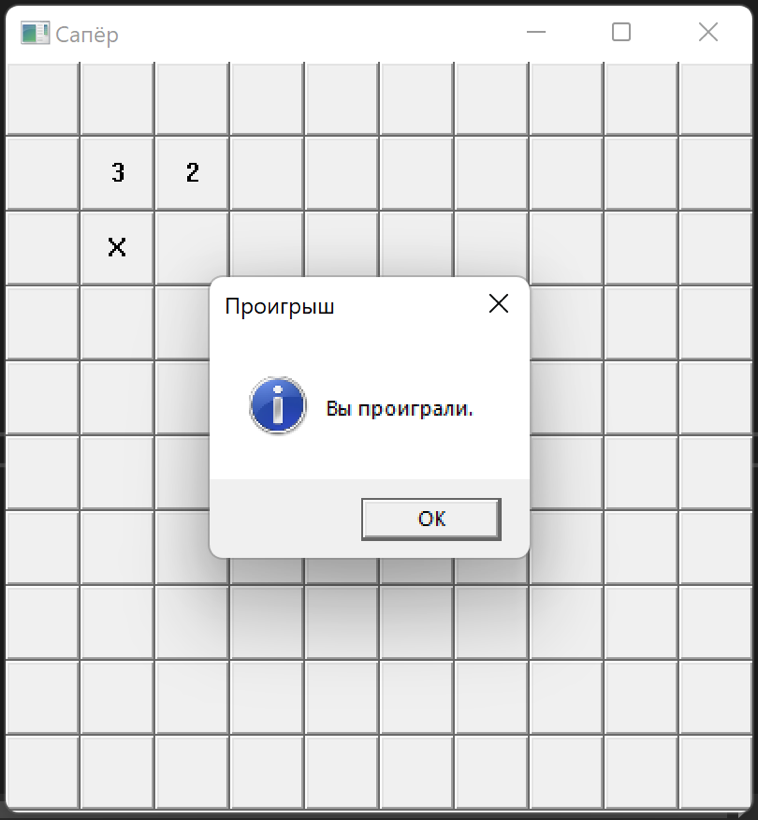


Рисунок 3.3 – Текстовое сообщение, информирующее о проигрыше

# 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] Окна на чистом WinAPI. Или просто о сложном [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/352096/>.

[3] Сообщения окна [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/window-messages>.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

Листинг 1 – Файл Lab1.h:

#pragma once

#include "Resource.h"

#include <string>

#include <windows.h>

#include "GameData.h"

Листинг 2 – Файл Lab1.cpp:

#ifndef UNICODE

#define UNICODE

#endif

#include "Lab1.h"

HINSTANCE hInst;

WCHAR szTitle[] = L"Сапёр";

WCHAR szWindowClass[] = L"MinesweeperClass";

const int cellSize = 40;

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

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow)

{

hInst = hInstance;

PlaceMines();

WNDCLASSEXW wcex = { 0 };

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.hInstance = hInstance;

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

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

wcex.lpszClassName = szWindowClass;

if (!RegisterClassExW(&wcex))

{

return 1;

}

HWND hWnd = CreateWindowW(

szWindowClass,

szTitle,

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT,

CW\_USEDEFAULT,

415,

440,

nullptr,

nullptr,

hInstance,

nullptr

);

if (!hWnd)

{

return 2;

}

ShowWindow(hWnd, nCmdShow);

MSG msg;

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

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

{

switch (message)

{

case WM\_CREATE:

{

for (int y = 0; y < boardHeight; y++)

{

for (int x = 0; x < boardWidth; x++)

{

HWND hButton = CreateWindow(

L"BUTTON",

L"",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

x \* cellSize, y \* cellSize, cellSize, cellSize,

hWnd,

(HMENU)(y \* boardWidth + x + 1001),

hInst,

nullptr

);

InvalidateRect(hButton, NULL, TRUE);

}

}

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

if (wmId >= 1001 && wmId <= 1001 + boardHeight \* boardWidth)

{

int x = (wmId - 1001) % boardWidth;

int y = (wmId - 1001) / boardWidth;

if (GetKeyState(VK\_SHIFT) < 0)

{

MarkCell(x, y);

}

else

{

OpenCell(x, y);

if (CheckWinOrLose() == 0)

{

const wchar\_t\* message = L"Поздравляем, вы выиграли!";

MessageBox(NULL, message, L"Выигрыш!", MB\_ICONINFORMATION | MB\_OK);

}

else if (CheckWinOrLose() == 2)

{

const wchar\_t\* message = L"Вы проиграли.";

MessageBox(NULL, message, L"Проигрыш", MB\_ICONINFORMATION | MB\_OK);

}

}

InvalidateRect(hWnd, NULL, TRUE);

}

else

{

switch (wmId)

{

default:

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

}

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

for (int y = 0; y < boardHeight; y++)

{

for (int x = 0; x < boardWidth; x++)

{

int buttonId = 1001 + y \* boardWidth + x;

HWND hButton = GetDlgItem(hWnd, buttonId);

if (IsMarked(x, y))

{

SetWindowText(hButton, L"P");

}

else if (IsOpen(x, y))

{

if (HasMine(x, y))

{

SetWindowText(hButton, L"X");

}

else if (GetAdjacentMines(x, y) > 0)

{

std::wstring text = std::to\_wstring(GetAdjacentMines(x, y));

SetWindowText(hButton, text.c\_str());

}

else

{

SetWindowText(hButton, L"0");

}

}

else

{

SetWindowText(hButton, L"");

}

}

}

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

Листинг 3 – Файл GameData.h:

#pragma once

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <queue>

extern const int boardWidth;

extern const int boardHeight;

extern const int numberOfMines;

struct Cell

{

bool isOpen;

bool hasMine;

int adjacentMines;

bool isMarked;

Cell() : isOpen(false), hasMine(false), adjacentMines(0), isMarked(false) {}

};

void PlaceMines();

void CountAdjacentMines(int x, int y, Cell &cell);

bool HasMine(int x, int y);

bool IsOpen(int x, int y);

void OpenCell(int x, int y);

int GetAdjacentMines(int x, int y);

bool BorderCheck(int x, int y);

bool IsZeroAdjacentMines(int x, int y);

void OpenOneCell(int x, int y);

void MarkCell(int x, int y);

bool IsMarked(int x, int y);

int CheckWinOrLose();

Листинг 4 – Файл GameData.cpp:

#include "GameData.h"

const int boardWidth = 10;

const int boardHeight = 10;

const int numberOfMines = 10;

Cell gameBoard[boardHeight][boardWidth];

int openCellsNumber = 0;

int markedCellsNumber = 0;

bool isLose = false;

void PlaceMines()

{

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

int minesPlaced = 0;

while (minesPlaced < numberOfMines)

{

int x = std::rand() % boardWidth;

int y = std::rand() % boardHeight;

if (!gameBoard[y][x].hasMine)

{

gameBoard[y][x].hasMine = true;

++minesPlaced;

}

}

for (int y = 0; y < boardHeight; ++y)

{

for (int x = 0; x < boardWidth; ++x)

{

CountAdjacentMines(x, y, gameBoard[y][x]);

}

}

}

void CountAdjacentMines(int x, int y, Cell &cell)

{

int count = 0;

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

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

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

{

int newX = x + dx[i];

int newY = y + dy[i];

if (newX >= 0 && newX < boardWidth && newY >= 0 && newY < boardHeight)

{

if (gameBoard[newY][newX].hasMine)

{

count++;

}

}

}

cell.adjacentMines = count;

}

bool HasMine(int x, int y)

{

return gameBoard[y][x].hasMine;

}

bool IsOpen(int x, int y)

{

return gameBoard[y][x].isOpen;

}

int GetAdjacentMines(int x, int y)

{

return gameBoard[y][x].adjacentMines;

}

bool IsZeroAdjacentMines(int x, int y)

{

if (GetAdjacentMines(x, y) > 0)

return false;

else

return true;

}

void OpenOneCell(int x, int y)

{

if (!IsOpen(x, y))

{

++openCellsNumber;

gameBoard[y][x].isOpen = true;

}

if (IsMarked(x, y))

{

gameBoard[y][x].isMarked = false;

--markedCellsNumber;

}

}

bool BorderCheck(int x, int y)

{

if (x >= 0 && x < boardWidth && y >= 0 && y < boardHeight)

return true;

else

return false;

}

void OpenCell(int x, int y)

{

std::queue<std::pair<int, int>> cellQueue;

if (!gameBoard[y][x].isOpen && gameBoard[y][x].hasMine)

{

gameBoard[y][x].isOpen = true;

if (IsMarked(x, y))

{

gameBoard[y][x].isMarked = false;

--markedCellsNumber;

}

isLose = true;

return;

}

if (gameBoard[y][x].isOpen)

{

return;

}

cellQueue.push(std::make\_pair(x, y));

while (!cellQueue.empty())

{

int currentX = cellQueue.front().first;

int currentY = cellQueue.front().second;

cellQueue.pop();

OpenOneCell(currentX, currentY);

gameBoard[currentY][currentX].isMarked = false;

if (IsZeroAdjacentMines(currentX, currentY))

{

for (int dx = currentX - 1; dx < currentX + 2; ++dx)

{

for (int dy = currentY - 1; dy < currentY + 2; ++dy)

{

if (BorderCheck(dx, dy) && !HasMine(dx, dy) && IsZeroAdjacentMines(dx, dy) && !IsOpen(dx, dy))

{

cellQueue.push(std::make\_pair(dx, dy));

}

else if (BorderCheck(dx, dy) && !HasMine(dx, dy) && !IsZeroAdjacentMines(dx, dy) && !IsOpen(dx, dy))

{

OpenOneCell(dx, dy);

}

}

}

}

}

}

void MarkCell(int x, int y)

{

if (!IsOpen(x, y) && IsMarked(x, y))

{

gameBoard[y][x].isMarked = false;

++markedCellsNumber;

}

else if (!IsOpen(x, y) && !IsMarked(x, y))

{

gameBoard[y][x].isMarked = true;

--markedCellsNumber;

}

}

bool IsMarked(int x, int y)

{

return gameBoard[y][x].isMarked;

}

int CheckWinOrLose()

{

if (isLose)

{

return 2;

}

else if (markedCellsNumber <= numberOfMines && openCellsNumber == boardWidth \* boardHeight - numberOfMines)

{

return 0;

}

return 1;

}