**Source Code**

***Creating a window***

// Win32Project2.cpp : Defines the entry point for the application.

//

#include "stdafx.h"

#include "Win32Project2.h"

#include <windowsx.h>

#include <TIME.H>

#define MAX\_LOADSTRING 100

// Global Variables:

HINSTANCE hInst; // current instance

WCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

WCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

// Initialize global strings

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

LoadStringW(hInstance, IDC\_WIN32PROJECT2, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_WIN32PROJECT2));

MSG msg;

// Main message loop:

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

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

//

// FUNCTION: MyRegisterClass()

//

// PURPOSE: Registers the window class.

//

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, MAKEINTRESOURCE(IDI\_WIN32PROJECT2));

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

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

wcex.hbrBackground = (HBRUSH)(GetStockObject(GRAY\_BRUSH));

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_WIN32PROJECT2);

wcex.lpszClassName = szWindowClass;

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

return RegisterClassExW(&wcex);

}

//

// FUNCTION: InitInstance(HINSTANCE, int)

//

// PURPOSE: Saves instance handle and creates main window

//

// COMMENTS:

//

// In this function, we save the instance handle in a global variable and

// create and display the main program window.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Store instance handle in our global variable

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// PURPOSE: Processes messages for the main window.

//

// WM\_COMMAND - process the application menu

// WM\_PAINT - Paint the main window

// WM\_DESTROY - post a quit message and return

//

***Creating Variable for the game***

//Global variables

const int CELL\_SIZE = 100;

HBRUSH hbr1, hbr2, hbr3, hbr4;

int board[4] = { 0,1,2,3};

int playerTurn = 0;

int mode = 6;

int gameBoard[25] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };

int winner = 0;

int wins[4];

int player1 = 0;

int player2 = 0;

int guest = 1;

WCHAR \* c;

const WCHAR \* select[4];

/\*

Chart

00-01-02-03-04

05-06-07-08-09

10-11-12-13-14

15-16-17-18-19

20-21-22-23-24

\*/

int cells[] = { 0,1,2,3, 1,2,3,4, 5,6,7,8, 6,7,8,9, 10,11,12,13, 11,12,13,14, 15,16,17,18, 16,17,18,19, 20,21,22,23, 21,22,23,24,

0,5,10,15, 5,10,15,20, 1,6,11,16, 6,11,16,21, 2,7,12,17, 7,12,17,22, 3,8,13,18, 8,13,18,23, 4,9,14,19, 9,14,19,24,

0,6,12,18, 6,12,18,24, 1,7,13,19, 5,11,17,23, 4,8,12,16, 8,12,16,20, 9,13,17,21, 3,7,11,15};

WCHAR \* szPlayer1[10];

WCHAR \* szPlayer2[10];

const WCHAR szPlayer3[] = L"AI";

int index;

***Creating Fucntions for the game***

void time(HWND hWnd, HDC hdc)

{

clock\_t endwait;

endwait = clock() + 1 \* CLOCKS\_PER\_SEC;

while (clock() < endwait) {}

}

BOOL GetGameBoard(HWND hWnd, RECT \* pRect)

{

RECT rc;

if (GetClientRect(hWnd, &rc))

{

int width = rc.right - rc.left;

int height = rc.bottom - rc.top;

if (mode >= 6)

{

pRect->left = (width - CELL\_SIZE \* 3) / 2;

pRect->top = (height - CELL\_SIZE \* 3) / 2;

pRect->right = pRect->left + CELL\_SIZE \* 3;

pRect->bottom = pRect->top + CELL\_SIZE \* 3;

}

else if (mode >= 4)

{

pRect->left = (width - CELL\_SIZE \* 4) / 2;

pRect->top = (height - CELL\_SIZE \* 4) / 2;

pRect->right = pRect->left + CELL\_SIZE \* 4;

pRect->bottom = pRect->top + CELL\_SIZE \* 4;

}

else

{

pRect->left = (width - CELL\_SIZE \* 5) / 2;

pRect->top = (height - CELL\_SIZE \* 5) / 2;

pRect->right = (pRect->left + CELL\_SIZE \* 5);

pRect->bottom = (pRect->top + CELL\_SIZE \* 5);

}

return TRUE;

}

SetRectEmpty(pRect);

return FALSE;

}

void Drawlines(HDC hdc, int x1, int y1, int x2, int y2)

{

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, x2, y2);

}

int GetCellNumberFromPoint(HWND hwnd, int x, int y)

{

POINT pt = { x, y };

RECT rc;

if (GetGameBoard(hwnd, &rc))

{

if (PtInRect(&rc, pt))

{

//use clicks in the board

x = pt.x - rc.left;

y = pt.y - rc.top;

int column = x / CELL\_SIZE;

int row = y / CELL\_SIZE;

if (mode >= 4)

{

int row = (y / CELL\_SIZE);

return (row);

}

return (column + row \* 5);

}

}

//outside tic-tac-toe board

return -1;

}

BOOL GetCellRect(HWND hWnd, int index, RECT \*pRect)

{

RECT rcBoard;

SetRectEmpty(pRect);

if (index < 0 || index > 24)

return FALSE;

if (GetGameBoard(hWnd, &rcBoard))

{

int y, x;

if (mode == 6)

{

y = index;

x = index / 3;

}

else if (mode >= 4)

{

y = index;

x = index / 4;

}

else

{

y = index / 5;

x = index % 5;

}

if (mode == 6)

{

pRect->left = rcBoard.left + x \* CELL\_SIZE + 10;

pRect->top = rcBoard.top + y \* CELL\_SIZE + 10;

pRect->right = pRect->left + CELL\_SIZE + 180;

pRect->bottom = pRect->top + CELL\_SIZE - 20;

}

else if (mode >= 4)

{

pRect->left = rcBoard.left + x \* CELL\_SIZE + 10;

pRect->top = rcBoard.top + y \* CELL\_SIZE + 10;

pRect->right = pRect->left + CELL\_SIZE + 280;

pRect->bottom = pRect->top + CELL\_SIZE - 20;

}

else

{

pRect->left = rcBoard.left + x \* CELL\_SIZE + 10;

pRect->top = rcBoard.top + y \* CELL\_SIZE + 10;

pRect->right = pRect->left + CELL\_SIZE - 20;

pRect->bottom = pRect->top + CELL\_SIZE - 20;

}

return TRUE;

}

return FALSE;

}

//returns the winner

int Winner(int wins[4])

{

//row, verticals, columns

for (int i = 0; i < ARRAYSIZE(cells); i += 4)

{

if (0 != gameBoard[cells[i]] && gameBoard[cells[i]] == gameBoard[cells[i + 1]] && gameBoard[cells[i]] == gameBoard[cells[i + 2]] && gameBoard[cells[i]] == gameBoard[cells[i + 3]])

{

wins[0] = cells[i];

wins[1] = cells[i + 1];

wins[2] = cells[i + 2];

return gameBoard[cells[i]];

}

}

for (int i = 0; i < ARRAYSIZE(gameBoard); ++i)

{

if (0 == gameBoard[i])

return 0;

}

return 3;

}

void ShowTurn(HWND hWnd, HDC hdc)

{

const WCHAR \* TurnText = NULL;

switch (winner)

{

case 0:

if (mode == 0)

{

TurnText = (playerTurn == 1) ? szPlayer1[player1] : szPlayer2[player2];

}

if (mode != 0)

{

TurnText = (playerTurn == 1) ? szPlayer1[player1] : szPlayer3;

}

break;

case 1:

TurnText = L"Player 1 is the winnner!";

break;

case 2:

TurnText = L"Player 2 is the winnner!";

break;

case 3:

TurnText = L"It's a draw!";

break;

case 4:

TurnText = L"AI is the winnner!";

break;

}

RECT rc;

if (NULL != TurnText && GetClientRect(hWnd, &rc))

{

rc.top = rc.bottom - 48;

FillRect(hdc, &rc, (HBRUSH)GetStockObject(GRAY\_BRUSH));

SetTextColor(hdc, RGB(255, 255, 255));

SetBkMode(hdc, TRANSPARENT);

DrawText(hdc, TurnText, lstrlen(TurnText), &rc, DT\_CENTER);

}

}

//Player and AI choice

void Moves(HWND hWnd, HDC hdc, int xPos, int yPos)

{

if (mode == 0)

{

//calling function for button

index = GetCellNumberFromPoint(hWnd, xPos, yPos);

//print out Value of cell clicked on

if (NULL != hdc)

{

/\*WCHAR temp[100];

wsprintf(temp, L"Index = %d", index);

TextOut(hdc, xPos, yPos, temp, lstrlen(temp));\*/

//get cell demensions

if (index != -1)

{

RECT rcCell;

//cell is taken no other player can take over

if ((0 == gameBoard[index]) && GetCellRect(hWnd, index, &rcCell))

{

gameBoard[index] = playerTurn;

FillRect(hdc, &rcCell, (playerTurn == 2) ? hbr2 : hbr1);

//FillRect(hdc, &rcCell, hbr1);

winner = Winner(wins);

if (winner == 1 || winner == 2)

{

MessageBox(hWnd, (winner == 1) ? L"Player 1 is the winnner!" : L"Player 2 is the winner!", L"You Win!", MB\_OK | MB\_ICONINFORMATION);

playerTurn = 0;

}

else if (3 == winner)

{

MessageBox(hWnd, L"No one wins!", L"It's a draw", MB\_OK | MB\_ICONEXCLAMATION);

playerTurn = 0;

}

else if (0 == winner)

{

playerTurn = (playerTurn == 1) ? 2 : 1;

}

ShowTurn(hWnd, hdc);

}

}

ReleaseDC(hWnd, hdc);

}

}

}

void Move(HWND hWnd, HDC hdc, int xPos, int yPos)

{

if (mode >= 1 && playerTurn == 1)

{

//calling function for button

index = GetCellNumberFromPoint(hWnd, xPos, yPos);

//print out Value of cell clicked on

if (NULL != hdc)

{

/\*WCHAR temp[100];

wsprintf(temp, L"Index = %d", index);

TextOut(hdc, xPos, yPos, temp, lstrlen(temp));\*/

//get cell demensions

if (index != -1)

{

RECT rcCell;

//cell is taken no other player can take over

if ((0 == gameBoard[index]) && GetCellRect(hWnd, index, &rcCell))

{

gameBoard[index] = playerTurn;

FillRect(hdc, &rcCell, hbr1);

winner = Winner(wins);

if (winner == 1)

{

MessageBox(hWnd, L"Player 1 is the winnner!", L"You Win!", MB\_OK | MB\_ICONINFORMATION);

playerTurn = 0;

}

else if (3 == winner)

{

MessageBox(hWnd, L"No one wins!", L"It's a draw", MB\_OK | MB\_ICONEXCLAMATION);

playerTurn = 0;

}

else if (0 == winner)

{

playerTurn = (playerTurn == 1) ? 4 : 1;

}

ShowTurn(hWnd, hdc);

}

}

ReleaseDC(hWnd, hdc);

}

}

}

int modes(int &index)

{

//stoped here!

int winner = 25;

int loser1 = 25;

int loser2 = 25;

int opt1 = 25;

int opt2 = 25;

for (int i = 0; i < ARRAYSIZE(cells); i += 4)

{

if (mode > 0)

{

if (mode > 1)

{

if (mode > 2)

{

//block hard

if (1 == gameBoard[cells[i]] && 1 == gameBoard[cells[i + 1]] && 0 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 3]])

loser2 = cells[i + 2];

else if (1 == gameBoard[cells[i]] && 1 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 1]] && 0 == gameBoard[cells[i + 3]])

loser2 = cells[i + 1];

else if (1 == gameBoard[cells[i]] && 1 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 1]])

loser2 = cells[i + 2];

else if (1 == gameBoard[cells[i + 1]] && 1 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i]] && 0 == gameBoard[cells[i + 3]])

loser2 = cells[i];

else if (1 == gameBoard[cells[i + 1]] && 1 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i]] && 0 == gameBoard[cells[i + 2]])

loser2 = cells[i];

else if (1 == gameBoard[cells[i + 2]] && 1 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i]] && 0 == gameBoard[cells[i + 1]])

loser2 = cells[i];

}

//block medium

if (1 == gameBoard[cells[i]] && 1 == gameBoard[cells[i + 2]] && 1 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 1]])

loser1 = cells[i + 1];

else if (1 == gameBoard[cells[i]] && 1 == gameBoard[cells[i + 1]] && 1 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 2]])

loser1 = cells[i + 2];

else if (1 == gameBoard[cells[i]] && 1 == gameBoard[cells[i + 1]] && 1 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 3]])

loser1 = cells[i + 3];

else if (1 == gameBoard[cells[i + 1]] && 1 == gameBoard[cells[i + 2]] && 1 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i]])

loser1 = cells[i];

//medium options

else if (4 == gameBoard[cells[i]] && 0 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 1]])

opt2 = cells[i + 1];

else if (0 == gameBoard[cells[i]] && 0 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 3]] && 4 == gameBoard[cells[i + 1]])

opt2 = cells[i];

else if (0 == gameBoard[cells[i]] && 4 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 1]])

opt2 = cells[i];

else if (0 == gameBoard[cells[i]] && 0 == gameBoard[cells[i + 2]] && 4 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 1]])

opt2 = cells[i];

//options to process to win

else if (4 == gameBoard[cells[i]] && 4 == gameBoard[cells[i + 1]] && 0 == gameBoard[cells[i + 2]] && 1 != gameBoard[cells[i + 3]])

opt1 = cells[i + 2];

else if (4 == gameBoard[cells[i]] && 4 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 1]] && 1 != gameBoard[cells[i + 3]])

opt1 = cells[i + 1];

else if (4 == gameBoard[cells[i]] && 4 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 2]] && 1 != gameBoard[cells[i + 1]])

opt1 = cells[i + 2];

else if (4 == gameBoard[cells[i + 1]] && 4 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i]] && 1 != gameBoard[cells[i + 3]])

opt1 = cells[i];

else if (4 == gameBoard[cells[i + 1]] && 4 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i]] && 1 != gameBoard[cells[i + 2]])

opt1 = cells[i];

else if (4 == gameBoard[cells[i + 2]] && 4 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i]] && 1 != gameBoard[cells[i + 1]])

opt1 = cells[i];

}

//win

if (4 == gameBoard[cells[i]] && 4 == gameBoard[cells[i + 2]] && 4 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 1]])

winner = cells[i + 1];

else if (4 == gameBoard[cells[i]] && 4 == gameBoard[cells[i + 1]] && 4 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i + 2]])

winner = cells[i + 2];

else if (4 == gameBoard[cells[i]] && 4 == gameBoard[cells[i + 1]] && 4 == gameBoard[cells[i + 2]] && 0 == gameBoard[cells[i + 3]])

winner = cells[i + 3];

else if (4 == gameBoard[cells[i + 1]] && 4 == gameBoard[cells[i + 2]] && 4 == gameBoard[cells[i + 3]] && 0 == gameBoard[cells[i]])

winner = cells[i];

// easy options

else

{

opt2 = rand() % 25;

while (1 == gameBoard[opt2] || 4 == gameBoard[opt2])

opt2 = rand() % 25;

}

}

}

if (winner != 25)

index = winner;

else if (loser1 != 25)

index = loser1;

else if (loser2 != 25)

index = loser2;

else if (opt1 != 25)

index = opt1;

else if (opt2 != 25)

index = opt2;

//working on this

return index;

}

void PCMoves(HWND hWnd, HDC hdc)

{

if (4 == playerTurn)

{

//calling function for button

//print out Value of cell clicked on

HDC hdc = GetDC(hWnd);

if (NULL != hdc)

{

/\*WCHAR temp[100];

wsprintf(temp, L"Index = %d", index);

TextOut(hdc, xPos, yPos, temp, lstrlen(temp));\*/

//get cell demensions

RECT rcCell;

//AI

modes(index);

//cell is taken no other player can take over

if ((0 == gameBoard[index]) && GetCellRect(hWnd, index, &rcCell))

{

gameBoard[index] = playerTurn;

FillRect(hdc, &rcCell, hbr3);

winner = Winner(wins);

if (winner == 4)

{

MessageBox(hWnd, L"AI is the winner!", L"You Lose!", MB\_OK | MB\_ICONINFORMATION);

playerTurn = 0;

}

else if (3 == winner)

{

MessageBox(hWnd, L"No one wins!", L"It's a draw", MB\_OK | MB\_ICONEXCLAMATION);

playerTurn = 0;

}

else if (0 == winner)

{

playerTurn = (playerTurn == 1) ? 4 : 1;

}

ShowTurn(hWnd, hdc);

}

}

ReleaseDC(hWnd, hdc);

}

}

void Horizon(HDC hdc, int x1, int y1, int x2, int y2)

{

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, x2, y2);

}

void reset(HWND hWnd, HDC hdc)

{

if (mode >= 4)

playerTurn = 0;

else

{

int ans = MessageBox(hWnd, L"Would you like to make the first move?", L"Player 1", MB\_YESNO | MB\_ICONQUESTION);

if (IDNO == ans)

{

if (mode == 0)

playerTurn = 2;

else

playerTurn = 4;

}

else

playerTurn = 1;

}

winner = 0;

ZeroMemory(gameBoard, sizeof(gameBoard));

InvalidateRect(hWnd, NULL, TRUE);

UpdateWindow(hWnd);

if (mode > 0)

{

PCMoves(hWnd, hdc);

}

}

void selection(HWND hWnd, HDC hdc, int xPos, int yPos)

{

RECT rcCell;

index = GetCellNumberFromPoint(hWnd, xPos, yPos);

if (NULL != hdc)

{

if ((0 == board[index]) && GetCellRect(hWnd, index, &rcCell))

{

FillRect(hdc, &rcCell, hbr1);

time(hWnd, hdc);

if (mode == 6)

{

CreateWindowEx(WS\_EX\_CLIENTEDGE, L"edit", L"User\_Name",

WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP | WS\_BORDER | ES\_LEFT,

CW\_USEDEFAULT, CW\_USEDEFAULT, 200, 24, // x, y, w, h

hWnd, (HMENU)(101),

(HINSTANCE) GetWindowLong(hWnd, GWL\_HINSTANCE), NULL);

CreateWindowEx(BS\_PUSHBUTTON, L"Button", L"Enter",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER,

200, 34, 200, 34,

hWnd, NULL, NULL, NULL);

mode = 5;

}

else if (mode == 5)

{

int ret = MessageBox(hWnd, L"Player 2 would you like to Sing In?", L"Sing In", MB\_YESNO | MB\_ICONQUESTION);

if (IDYES == ret)

{

szPlayer1[player1] = szPlayer2[player2];

}

else

{

szPlayer1[player1] = szPlayer2[player2];

szPlayer2[player2] = L"Guest ";

}

mode = 0;

}

else

mode = 1;

reset( hWnd, hdc);

}

else if ((1 == board[index]) && GetCellRect(hWnd, index, &rcCell))

{

FillRect(hdc, &rcCell, hbr2);

time(hWnd, hdc);

if (mode == 6)

{

szPlayer2[player2] = L"Guest";

mode = 5;

}

else if (mode == 5)

mode = 4;

else

mode = 2;

reset(hWnd, hdc);

}

else if ((2 == board[index]) && GetCellRect(hWnd, index, &rcCell))

{

FillRect(hdc, &rcCell, hbr4);

time(hWnd, hdc);

if (mode == 6)

DestroyWindow(hWnd);

else if (mode == 4)

mode = 3;

reset(hWnd, hdc);

}

else if ((3 == board[index]) && GetCellRect(hWnd, index, &rcCell))

{

FillRect(hdc, &rcCell, hbr3);

time(hWnd, hdc);

if (mode == 5)

mode = 6;

else

mode = 5;

reset(hWnd, hdc);

}

}

}

***Drawing the Game in the window***

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

{

switch (message)

{

//create a brush to user as player

case WM\_CREATE:

{

hbr1 = CreateSolidBrush(RGB(255, 0, 0));

hbr2 = CreateSolidBrush(RGB(0, 0, 255));

hbr3 = CreateSolidBrush(RGB(0, 225, 0));

hbr4 = CreateSolidBrush(RGB(200, 200, 200));

}

case WM\_COMMAND:

{

HDC hdc = GetDC(hWnd);

int wmId = LOWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case ID\_NEWGAME:

{

int ret = MessageBox(hWnd, L"Are you sure you want to start a new game?", L"New Game", MB\_YESNO | MB\_ICONQUESTION);

if (IDYES == ret)

{

mode = 0;

reset(hWnd, hdc);

}

}

break;

case ID\_SINGLEPLAYER\_EASY:

{

int ret = MessageBox(hWnd, L"Are you sure you want to start a new game?", L"New Game", MB\_YESNO | MB\_ICONQUESTION);

if (IDYES == ret)

{

mode = 1;

reset(hWnd, hdc);

}

}

break;

case ID\_SINGLEPLAYER\_MEDIUM:

{

int ret = MessageBox(hWnd, L"Are you sure you want to start a new game?", L"New Game", MB\_YESNO | MB\_ICONQUESTION);

if (IDYES == ret)

{

mode = 2;

reset(hWnd, hdc);

}

}

break;

case ID\_SINGLEPLAYER\_HARD:

{

int ret = MessageBox(hWnd, L"Are you sure you want to start a new game?", L"New Game", MB\_YESNO | MB\_ICONQUESTION);

if (IDYES == ret)

{

mode = 3;

reset(hWnd, hdc);

}

}

break;

case ID\_MAINMENU:

{

int ret = MessageBox(hWnd, L"Are you sure you want to go to the Main Menu?", L"Main Menu", MB\_YESNO | MB\_ICONQUESTION);

if (IDYES == ret)

{

mode = 6;

reset(hWnd, hdc);

}

}

break; case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

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

}

}

break;

***Button actions***

case WM\_LBUTTONDOWN:

{

int xPos = GET\_X\_LPARAM(lParam);

int yPos = GET\_Y\_LPARAM(lParam);

HDC hdc = GetDC(hWnd);

if (mode >= 4)

selection(hWnd, hdc, xPos, yPos);

else if (mode >= 1)

{

Move(hWnd, hdc, xPos, yPos);

}

else

{

Moves(hWnd, hdc, xPos, yPos);

}

}

break;

case WM\_LBUTTONUP:

{

int xPos = GET\_X\_LPARAM(lParam);

int yPos = GET\_Y\_LPARAM(lParam);

HDC hdc = GetDC(hWnd);

PCMoves(hWnd, hdc);

}

break;

case WM\_GETMINMAXINFO:

{

MINMAXINFO \* pMinMax = (MINMAXINFO\*)lParam;

pMinMax->ptMinTrackSize.x = (CELL\_SIZE \* 7);

pMinMax->ptMinTrackSize.y = (CELL\_SIZE \* 7);

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

RECT rc;

if (mode >= 4)

{

if (GetGameBoard(hWnd, &rc))

{

Rectangle(hdc, rc.left, rc.top, rc.right, rc.bottom);

}

for (int i = 0; i < 4; i++)

Horizon(hdc, rc.left, rc.top + CELL\_SIZE \*i, rc.right, rc.top + CELL\_SIZE \*i);

if (mode == 6)

{

select[1] = L"Sing In";

select[2] = L"Play as guest";

select[3] = L"Exit Game";

SetTextColor(hdc, RGB(0, 0, 0));

TextOut(hdc, rc.left + CELL\_SIZE \*1, rc.top + CELL\_SIZE \* .5, select[1], lstrlen(select[1]));

TextOut(hdc, rc.left + CELL\_SIZE \*1, rc.top + CELL\_SIZE \* 1.5, select[2], lstrlen(select[2]));

TextOut(hdc, rc.left + CELL\_SIZE \*1, rc.top + CELL\_SIZE \* 2.5, select[3], lstrlen(select[3]));

}

if (mode == 5)

{

select[1] = L"Multi-Player";

select[2] = L"Single Player";

select[3] = L"Score Player";

select[4] = L"Register";

SetTextColor(hdc, RGB(0, 0, 0));

TextOut(hdc, rc.left + CELL\_SIZE \*1.5 , rc.top + CELL\_SIZE \* .5 , select[1], lstrlen(select[1]));

TextOut(hdc, rc.left + CELL\_SIZE \*1.5, rc.top + CELL\_SIZE \* 1.5, select[2], lstrlen(select[2]));

TextOut(hdc, rc.left + CELL\_SIZE \*1.5, rc.top + CELL\_SIZE \* 2.5, select[3], lstrlen(select[3]));

TextOut(hdc, rc.left + CELL\_SIZE \*1.5, rc.top + CELL\_SIZE \* 3.5, select[4], lstrlen(select[4]));

}

if (mode == 4)

{

select[1] = L"Easy";

select[2] = L"Medium";

select[3] = L"Hard";

select[4] = L"Main Menu";

SetTextColor(hdc, RGB(0, 0, 0));

TextOut(hdc, rc.left + CELL\_SIZE \*1.5, rc.top + CELL\_SIZE \* .5, select[1], lstrlen(select[1]));

TextOut(hdc, rc.left + CELL\_SIZE \*1.5, rc.top + CELL\_SIZE \* 1.5, select[2], lstrlen(select[2]));

TextOut(hdc, rc.left + CELL\_SIZE \*1.5, rc.top + CELL\_SIZE \* 2.5, select[3], lstrlen(select[3]));

TextOut(hdc, rc.left + CELL\_SIZE \*1.5, rc.top + CELL\_SIZE \* 3.5, select[4], lstrlen(select[4]));

}

}

else

{

if (GetGameBoard(hWnd, &rc))

{

RECT rcClient;

if (GetClientRect(hWnd, &rcClient))

{

//display Player's name

SetBkMode(hdc, TRANSPARENT);

if (mode > 0)

{

SetTextColor(hdc, RGB(255, 0, 0));

TextOut(hdc, 18, 20, szPlayer1[player1], lstrlen(szPlayer1[player1]));

SetTextColor(hdc, RGB(0, 255, 0));

TextOut(hdc, rcClient.right - 80, 20, szPlayer3, ARRAYSIZE(szPlayer3));

}

if (mode == 0)

{

SetTextColor(hdc, RGB(255, 0, 0));

TextOut(hdc, 18, 20, szPlayer1[player1], lstrlen(szPlayer1[player1]));

SetTextColor(hdc, RGB(0, 0, 255));

TextOut(hdc, rcClient.right - 80, 20, szPlayer2[player2], lstrlen(szPlayer2[player2]));

}

ShowTurn(hWnd, hdc);

}

//Remove boarders

FillRect(hdc, &rc, (HBRUSH)GetStockObject(WHITE\_BRUSH));

//Rectangle(hdc, rc.left, rc.top, rc.right, rc.bottom);

}

for (int i = 1; i < 5; ++i)

{

//vertical lines

Drawlines(hdc, rc.left + CELL\_SIZE \* i, rc.top, rc.left + CELL\_SIZE \*i, rc.bottom);

//Horizantal line

Drawlines(hdc, rc.left, rc.top + CELL\_SIZE \* i, rc.right, rc.top + CELL\_SIZE \* i);

}

for (int i = 0; i < 25; i++)

{

RECT rcCell;

//action stays after re-size page

if (0 != gameBoard[i] && GetCellRect(hWnd, i, &rcCell) && mode == 0)

{

FillRect(hdc, &rcCell, (gameBoard[i] == 2) ? hbr2 : hbr1);

}

else if (0 != gameBoard[i] && GetCellRect(hWnd, i, &rcCell) && mode > 0)

{

FillRect(hdc, &rcCell, (gameBoard[i] == 4) ? hbr3 : hbr1);

}

}

}

// TODO: Add any drawing code that uses hdc here...

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

DeleteObject(hbr1);

DeleteObject(hbr2);

DeleteObject(hbr3);

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

// Message handler for about box.

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

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}