#include <afxwin.h>

#include <vector>

#include <algorithm>

// CWinApp :The base class from which you derive a Windows application object.

// CMyApp is derived from CWinApp and represents the application object

class CMyApp : public CWinApp

{

public:

// InitInstance is called to initialize the application

virtual BOOL InitInstance();

};

// CMainWindow is derived from CFrameWnd and represents the main window of the application

class CMainWindow : public CFrameWnd

{

public:

// Constructor for CMainWindow

CMainWindow();

protected:

// Message handlers for various Windows messages

afx\_msg void OnPaint();

afx\_msg void OnLButtonDown(UINT nFlags, CPoint point);

afx\_msg void OnRButtonDown(UINT nFlags, CPoint point);

afx\_msg BOOL OnEraseBkgnd(CDC\* pDC);

// Macro to declare a message map for this class

DECLARE\_MESSAGE\_MAP()

private:

std::vector<double> m\_values; // Vector of values used in this class

COLORREF m\_bgColor; // Background color for the window

};

// Handler for WM\_ERASEBKGND message - fills the client area with a solid color

BOOL CMainWindow::OnEraseBkgnd(CDC\* pDC) {

CRect rect;

GetClientRect(&rect);

pDC->FillSolidRect(&rect, m\_bgColor);

return TRUE;

}

CMyApp myApp; // Global instance of the application object

// Implementation of InitInstance - creates and shows the main window of the application

BOOL CMyApp::InitInstance()

{

m\_pMainWnd = new CMainWindow;

m\_pMainWnd->ShowWindow(m\_nCmdShow);

m\_pMainWnd->UpdateWindow();

return TRUE;

}

// Constructor for CMainWindow - creates a window and initializes member variables

CMainWindow::CMainWindow()

{

Create(NULL, \_T("MFC Application"));

m\_bgColor = RGB(135, 206, 250);

for (int i = 1; i <= 4; ++i)

m\_values.push\_back(static\_cast<double>(i));

}

// This macro defines a message map for the CMainWindow class

BEGIN\_MESSAGE\_MAP(CMainWindow, CFrameWnd)

// These macros specify that the corresponding Windows messages should be handled by member functions with the same name in the CMainWindow class

ON\_WM\_PAINT()

ON\_WM\_LBUTTONDOWN()

ON\_WM\_RBUTTONDOWN()

ON\_WM\_ERASEBKGND()

END\_MESSAGE\_MAP()

// This function is called when a window needs to be repainted

void CMainWindow::OnPaint()

{ // Create a device context object for painting in the client area of the window

CPaintDC dc(this);

// Retrieve the dimensions of the client area

CRect rect;

GetClientRect(&rect);

//Set background mode to transparent

dc.SetBkMode(TRANSPARENT);

// Calculate width and height of squares based on size of client area and number of values to display

int width = rect.Width() / (m\_values.size() \* 2 + 1);

int height = width;

// Reduce square size to 50% of its original size(if needed)

width \*= 1;

height \*= 1;

// Calculate spacing between squares

int spacing = width;

// Calculate total width of all squares and spacings

int totalWidth = m\_values.size() \* (width + spacing) - spacing;

for (size\_t i = 0; i < m\_values.size(); ++i)

{

CString str;

if (m\_values[i] == static\_cast<int>(m\_values[i]))

str.Format(\_T("%.0f"), m\_values[i]);

else

str.Format(\_T("%.2f"), m\_values[i]);

//str.Format(\_T("%.2f"), m\_values[i]);

// Calculate the rectangle where this value will be painted

CRect r(rect.left + (rect.Width() - totalWidth) / 2 + i \* (width + spacing), rect.top + (rect.Height() - height) / 2,

rect.left + (rect.Width() - totalWidth) / 2 + i \* (width + spacing) + width, rect.top + (rect.Height() - height) / 2 + height);

// Draw a rectangle around the value

dc.Rectangle(r);

// Fill the rectangle with a color

COLORREF color = RGB(255 \* i / m\_values.size(), 0, 255 \* (m\_values.size() - i - 1) / m\_values.size());

dc.FillSolidRect(r, color);

// Set text color to white

dc.SetTextColor(RGB(255, 255, 255));

// Paint the value onto the device context

dc.DrawText(str, &r, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);

}

}

void CMainWindow::OnRButtonDown(UINT nFlags, CPoint point) {

// Get the client rectangle of the window

CRect rect;

GetClientRect(&rect);

// Calculate the width and height of each square based on the number of values and the size of the client rectangle

int width = rect.Width() / (m\_values.size() \* 2 + 1);

int height = width;

// Reduce square size to 50% of its original size

width \*= 1;

height \*= 1;

// Calculate spacing between squares

int spacing = width;

// Initialize index to -1 (not found)

int index = -1;

// Calculate total width of all squares including spacing

int totalWidth = m\_values.size() \* (width + spacing) - spacing;

// Loop through all values to find which square was clicked on

for (size\_t i = 0; i < m\_values.size(); ++i) {

// Calculate left and right boundaries of current square

int left = rect.left + (rect.Width() - totalWidth) / 2 + i \* (width + spacing);

int right = left + width;

// Check if click was within current square's boundaries

if (point.x >= left && point.x <= right) {

index = i;

break;

}

}

// If a valid index was found, remove that value from the vector

if (index >= 0 && index < m\_values.size() && m\_values.size() > 1) {

m\_values.erase(m\_values.begin() + index);

}

// Redraw window

Invalidate();

}

void CMainWindow::OnLButtonDown(UINT nFlags, CPoint point) {

// Get the client rectangle of the window

CRect rect;

GetClientRect(&rect);

// Calculate the width of each value based on the number of values and the size of the client rectangle

int width = rect.Width() / m\_values.size();

// Calculate index based on x position of click and width of each value

int index = static\_cast<int>(std::floor(static\_cast<double>(point.x) / width + 0.5));

// Initialize average variable

double avg = 0;

if (index == 0) {

// If index is 0, insert new value at beginning of vector

m\_values.insert(m\_values.begin(), m\_values.front() - 1);

}

else if (index == m\_values.size()) {

// If index is equal to size of vector, insert new value at end of vector

m\_values.push\_back(m\_values.back() + 1);

}

else {

// Otherwise, calculate average between values at index-1 and index and insert new value at index

double avg = static\_cast<double>(m\_values[index - 1] + m\_values[index]) / 2;

m\_values.insert(m\_values.begin() + index, avg);

}

// Redraw window

Invalidate();

}