#define OEMRESOURCE

#include <Windows.h>

#include <commctrl.h>

#include <string.h>

#include "SoftwereDefenition.h"

int CALLBACK wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE, \_In\_ PWSTR, \_In\_ int nCmdShow) {

MSG msg{};

WNDCLASSEX wc = NewWindowClass

(

reinterpret\_cast<HBRUSH>(GetStockObject(WHITE\_BRUSH)),

LoadCursor(nullptr, IDC\_ARROW),

hInstance,

LoadIcon(nullptr, IDI\_ASTERISK),

L"RRT\_interactive",

SoftwareMainProc

);

if (!RegisterClassEx(&wc)) {

return EXIT\_FAILURE;

}

hwnd = CreateWindow(wc.lpszClassName, L"RRT algorithm",

WS\_OVERLAPPEDWINDOW

& ~WS\_THICKFRAME //WS\_THICKFRAME - has a sizing border

& ~WS\_MAXIMIZEBOX, //WS\_MAXIMIZEBOX - has a maximize button

(GetSystemMetrics(SM\_CXSCREEN) >> 1) - (WWidth >> 1), // x centering main window

(GetSystemMetrics(SM\_CYSCREEN) >> 1) - (WHight >> 1), // y centering main window

WWidth, WHight, nullptr, nullptr, wc.hInstance, nullptr);

if (hwnd == INVALID\_HANDLE\_VALUE) {

return EXIT\_FAILURE;

}

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return static\_cast<int>(msg.wParam); //or EXIT\_SUCCESS

}

inline WNDCLASSEX CALLBACK NewWindowClass(HBRUSH hBrush, HCURSOR hCursor, HINSTANCE hInstance, HICON hIcon, LPCWSTR name, WNDPROC Proc)

{

WNDCLASSEX wc{ sizeof(WNDCLASSEX) };

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = hBrush;

wc.hCursor = hCursor;

wc.hIcon = hIcon;

wc.hIconSm = LoadIcon(nullptr, IDI\_APPLICATION); //or IDI\_ASTERISK

wc.hInstance = hInstance;

wc.lpfnWndProc = Proc;

wc.lpszClassName = name;

wc.lpszMenuName = nullptr;

wc.style = 0x000;

return wc;

}

LRESULT CALLBACK SoftwareMainProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_COMMAND:

{

switch (wParam)

{

case AboutOnMenuClicked:

{

MessageBox(hwnd, L"Just chill", L"About was clicked!", MB\_OK);

}

return 0;

case SaveOnMenuClicked:

{

MessageBox(hwnd, L"Success!", L"Scene saving!", NULL);

}

return 0;

case LoadOnMenuClicked:

{

MessageBox(hwnd, L"Success!", L"Scene loading!", NULL);

}

return 0;

case SetObsPoint:

{

try

{

BOOL A\_xSuccess, A\_ySuccess, D\_xSuccess, D\_ySuccess;

long A\_x, A\_y, D\_x, D\_y;

A\_x = GetDlgItemInt(hwnd, ObsPointA\_xEdit, &A\_xSuccess, false);

A\_y = GetDlgItemInt(hwnd, ObsPointA\_yEdit, &A\_ySuccess, false);

D\_x = GetDlgItemInt(hwnd, ObsPointD\_xEdit, &D\_xSuccess, false);

D\_y = GetDlgItemInt(hwnd, ObsPointD\_yEdit, &D\_ySuccess, false);

if (A\_xSuccess && A\_ySuccess && D\_xSuccess && D\_ySuccess) {

ObsSet.push(new FilledRectangle(

hwnd, POINT{ A\_x, A\_y }, POINT{ D\_x,D\_y }, NULL, RGB(255, 116, 0))

);

canvas\_redraw();

}

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

return 0;

}

}

return 0;

case SetStartPoint:

{

try

{

SetGoalPoint(SetStartPoint);

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

return 0;

}

}

return 0;

case SetEndPoint:

{

try

{

SetGoalPoint(SetEndPoint);

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

return 0;

}

}

return 0;

case SetRemoveObsButton:

{

if (CapturesIsActive) {

CapturesIsActive = false;

}

ObsSet.~Set();

ObsSet = \*(new Set<class Rectangle>());

canvas\_redraw();

}

return 0;

default:

break;

}

}

return 0;

case WM\_PAINT:

{

BeginPaint(hwnd, &cnvsPS);

canvas\_redraw();

EndPaint(hwnd, &cnvsPS);

}

return 0;

case WM\_CREATE:

{

MainWndAddMenu(hwnd);

MainWndAddWidgets(hwnd);

canvas\_redraw();

}

return 0;

case WM\_DESTROY:

{

PostQuitMessage(EXIT\_SUCCESS);

ExitProcess(NULL);

}

return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

inline void MainWndAddMenu(HWND hwnd)

{

HMENU hMenu = CreateMenu();

HMENU SubMenu = CreateMenu();

AppendMenu(SubMenu, MF\_STRING, SaveOnMenuClicked, L"Save scene");

AppendMenu(SubMenu, MF\_STRING, LoadOnMenuClicked, L"Load scene");

AppendMenu(hMenu, MF\_POPUP, (UINT)SubMenu, L"File");

AppendMenu(hMenu, MF\_STRING, AboutOnMenuClicked, L"About");

SetMenu(hwnd, hMenu);

}

inline void MainWndAddWidgets(HWND hwnd)

{

RECT ClientRect{ 0 };

GetClientRect(hwnd, &ClientRect);

int CWidth = ClientRect.right;

int CHight = ClientRect.bottom;

cnvs = CreateWindowA(

"static", NULL,

WS\_VISIBLE | WS\_CHILD | WS\_CLIPCHILDREN | SS\_NOTIFY | SS\_WHITERECT,

0, 0, 3 \* CWidth / 4, CHight,

hwnd, nullptr, nullptr, nullptr

);

wpOrigCanvasProc = (WNDPROC) SetWindowLong(cnvs, GWL\_WNDPROC, (LONG)CanvasSubclassProc);

cnvsDC = GetDC(cnvs);

SetBkColor(GetDC(cnvs), RGB(255, 255, 255));

GetClientRect(cnvs, &cnvsRCT);

start.set\_hwnd(cnvs);

end.set\_hwnd(cnvs);

SetClassLong(cnvs, GCL\_HCURSOR, LONG(LoadCursor(NULL, IDC\_CROSS))); //IDC\_SIZEALL

// separator line

CreateWindowA(

"static", NULL, WS\_CHILD | WS\_VISIBLE | SS\_GRAYRECT,

3 \* CWidth / 4, 0, WSeparatorWidth, CHight,

hwnd, nullptr, nullptr, nullptr

);

// getting text metric to extract text hight

HDC hdc = GetDC(hwnd);

/\*SetBkColor(hdc, RGB(255, 255, 255));\*/

TEXTMETRICA tm{ 0 };

GetTextMetricsA(hdc, &tm);

//text-widget hight

auto THeight = tm.tmHeight + 2\*tm.tmExternalLeading;

CreateWindowA(

"static", "interactive mode:",

WS\_CHILD | WS\_VISIBLE ,

3 \* CWidth / 4 + WSeparatorWidth, 0, CWidth / 4 - WSeparatorWidth, THeight,

hwnd, nullptr, nullptr, nullptr

);

// creating radiobutton

CreateWindowA("button", "set start point",

WS\_CHILD | WS\_VISIBLE | BS\_AUTORADIOBUTTON,

3 \* CWidth / 4 + WSeparatorWidth, THeight,

CWidth / 4 - WSeparatorWidth, THeight,

hwnd, (HMENU)RadioButtonSP, nullptr, nullptr

);

CreateWindowA("button", "set end point",

WS\_CHILD | WS\_VISIBLE | BS\_AUTORADIOBUTTON,

3 \* CWidth / 4 + WSeparatorWidth, 2\*THeight,

CWidth / 4 - WSeparatorWidth, THeight,

hwnd, (HMENU)RadioButtonEP, nullptr, nullptr

);

CreateWindowA("button", "set obstacle",

WS\_CHILD | WS\_VISIBLE | BS\_AUTORADIOBUTTON,

3 \* CWidth / 4 + WSeparatorWidth, 3\*THeight,

CWidth / 4 - WSeparatorWidth, THeight,

hwnd, (HMENU)RadioButtonSetObs, nullptr, nullptr

);

CreateWindowA("button", "select obstacle",

WS\_CHILD | WS\_VISIBLE | BS\_AUTORADIOBUTTON,

3 \* CWidth / 4 + WSeparatorWidth, 4 \* THeight,

CWidth / 4 - WSeparatorWidth, THeight,

hwnd, (HMENU)RadioButtonSelectObs, nullptr, nullptr

);

CheckRadioButton(hwnd, RadioButtonSP, RadioButtonSetObs, RadioButtonSP);

CreateWindowA(

"static", NULL, WS\_CHILD | WS\_VISIBLE | SS\_GRAYRECT,

3 \* CWidth / 4 + WSeparatorWidth, 5 \* THeight,

CWidth / 4 - WSeparatorWidth, WSeparatorWidth,

hwnd, nullptr, nullptr, nullptr

);

//CreateWindowSeparator(4 \* THeight, CWidth);

// start point widget creating:

CreateWindowA(

"static", "start point: ",

WS\_CHILD | WS\_VISIBLE,

3 \* CWidth / 4 + WSeparatorWidth, 5 \* THeight + WSeparatorWidth,

CWidth / 4 - WSeparatorWidth, THeight,

hwnd, nullptr, nullptr, nullptr

);

StartPointX = CreateWindowA(

"edit", NULL,

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

3 \* CWidth / 4 + WSeparatorWidth, 6 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 3, THeight,

hwnd, (HMENU)StartPointXEdit, nullptr, nullptr

);

SendMessage(StartPointX, EM\_SETCUEBANNER, 0, (LPARAM)L"x pos");

StartPointY = CreateWindowA(

"edit", NULL,

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

3 \* CWidth / 4 + WSeparatorWidth + (CWidth / 4 - WSeparatorWidth) / 3,

6 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 3, THeight,

hwnd, (HMENU)StartPointYEdit, nullptr, nullptr

);

SendMessage(StartPointY, EM\_SETCUEBANNER, 0, (LPARAM)L"y pos");

CreateWindowA(

"button", "set",

WS\_CHILD | WS\_VISIBLE,

3 \* CWidth / 4 + WSeparatorWidth + 2 \* (CWidth / 4 - WSeparatorWidth) / 3,

6 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 4, THeight,

hwnd, (HMENU)SetStartPoint, nullptr, nullptr

);

// end point widget creating :

CreateWindowA(

"static", "end point: ",

WS\_CHILD | WS\_VISIBLE,

3 \* CWidth / 4 + WSeparatorWidth, 7 \* THeight + WSeparatorWidth,

CWidth / 4 - WSeparatorWidth, THeight,

hwnd, nullptr, nullptr, nullptr

);

EndPointX = CreateWindowA(

"edit", NULL,

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

3 \* CWidth / 4 + WSeparatorWidth, 8 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 3, THeight,

hwnd, (HMENU)EndPointXEdit, nullptr, nullptr

);

SendMessage(EndPointX, EM\_SETCUEBANNER, 0, (LPARAM)L"x pos");

EndPointY = CreateWindowA(

"edit", NULL,

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

3 \* CWidth / 4 + WSeparatorWidth + (CWidth / 4 - WSeparatorWidth) / 3,

8 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 3, THeight,

hwnd, (HMENU)EndPointYEdit, nullptr, nullptr

);

SendMessage(EndPointY, EM\_SETCUEBANNER, 0, (LPARAM)L"y pos");

CreateWindowA(

"button", "set",

WS\_CHILD | WS\_VISIBLE,

3 \* CWidth / 4 + WSeparatorWidth + 2 \* (CWidth / 4 - WSeparatorWidth) / 3,

8 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 4, THeight,

hwnd, (HMENU)SetEndPoint, nullptr, nullptr

);

// obstacle widget creating:

CreateWindowA(

"static", "obstacle: ",

WS\_CHILD | WS\_VISIBLE,

3 \* CWidth / 4 + WSeparatorWidth, 9 \* THeight + WSeparatorWidth,

CWidth / 4 - WSeparatorWidth, THeight,

hwnd, nullptr, nullptr, nullptr

);

ObsPointA\_x = CreateWindowA(

"edit", NULL,

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

3 \* CWidth / 4 + WSeparatorWidth,

10 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 3, THeight,

hwnd, (HMENU)ObsPointA\_xEdit, nullptr, nullptr

);

SendMessage(ObsPointA\_x, EM\_SETCUEBANNER, 0, (LPARAM)L"A\_x pos");

ObsPointA\_y = CreateWindowA(

"edit", NULL,

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

3 \* CWidth / 4 + WSeparatorWidth + (CWidth / 4 - WSeparatorWidth) / 3,

10 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 3, THeight,

hwnd, (HMENU)ObsPointA\_yEdit, nullptr, nullptr

);

SendMessage(ObsPointA\_y, EM\_SETCUEBANNER, 0, (LPARAM)L"A\_y pos");

ObsPointD\_x = CreateWindowA(

"edit", NULL,

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

3 \* CWidth / 4 + WSeparatorWidth,

11 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 3, THeight,

hwnd, (HMENU)ObsPointD\_xEdit, nullptr, nullptr

);

SendMessage(ObsPointD\_x, EM\_SETCUEBANNER, 0, (LPARAM)L"D\_x pos");

ObsPointD\_y = CreateWindowA(

"edit", NULL,

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

3 \* CWidth / 4 + WSeparatorWidth + (CWidth / 4 - WSeparatorWidth) / 3,

11 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 3, THeight,

hwnd, (HMENU)ObsPointD\_yEdit, nullptr, nullptr

);

SendMessage(ObsPointD\_y, EM\_SETCUEBANNER, 0, (LPARAM)L"D\_y pos");

CreateWindowA(

"button", "set",

WS\_CHILD | WS\_VISIBLE,

3 \* CWidth / 4 + WSeparatorWidth + 2 \* (CWidth / 4 - WSeparatorWidth) / 3,

10 \* THeight + WSeparatorWidth,

(CWidth / 4 - WSeparatorWidth) / 4, 2 \* THeight,

hwnd, (HMENU)SetObsPoint, nullptr, nullptr

);

CreateWindowA(

"static", NULL, WS\_CHILD | WS\_VISIBLE | SS\_GRAYRECT,

3 \* CWidth / 4 + WSeparatorWidth,

12 \* THeight + 1.5 \* WSeparatorWidth,

CWidth / 4 - WSeparatorWidth, WSeparatorWidth,

hwnd, nullptr, nullptr, nullptr

);

//CreateWindowSeparator(11 \* THeight + 1.5\*WSeparatorWidth, CWidth);

CreateWindowA(

"button", "remove all obstacles",

WS\_CHILD | WS\_VISIBLE,

3 \* CWidth / 4 + WSeparatorWidth,

12 \* THeight + 2.5 \* WSeparatorWidth,

CWidth / 4 - WSeparatorWidth, THeight,

hwnd, (HMENU)SetRemoveObsButton, nullptr, nullptr

);

CreateWindowA(

"static", NULL, WS\_CHILD | WS\_VISIBLE | SS\_GRAYRECT,

3 \* CWidth / 4 + WSeparatorWidth,

13 \* THeight + 2.5 \* WSeparatorWidth,

CWidth / 4 - WSeparatorWidth, WSeparatorWidth,

hwnd, nullptr, nullptr, nullptr

);

//CreateWindowSeparator(12 \* THeight + 2.5 \* WSeparatorWidth, CWidth);

}

LRESULT CALLBACK CanvasSubclassProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

//SendMessage(cptrA, WM\_NCPAINT, 1, 0);

try

{

switch (uMsg)

{

case WM\_MOUSEMOVE:

{

if (ObsSet.is\_empty())

return 0;

if (SetObsLButtonPressed) {

try

{

int x\_2 = GET\_X\_LPARAM(lParam);

int y\_2 = GET\_Y\_LPARAM(lParam);

//POINT A = GetLastObs->get\_a();

POINT A = GetLastObs->get\_a();

POINT B({ x\_2, A.y });

GetLastObs->set\_i\_point(1, B);

POINT C({ x\_2, y\_2 });

GetLastObs->set\_i\_point(2, C);

POINT D({ A.x, y\_2 });

GetLastObs->set\_i\_point(3, D);

canvas\_redraw();

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

}

/\*catch (...) {}\*/

}

}

return 0;

case WM\_LBUTTONDOWN:

{

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

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

if (CapturesIsActive) {

Circle\* SelectedCptr = CptrSetColision(x,y);

if (SelectedCptr) {

SelCptrLButtonPressed = true;

}

else {

CapturesIsActive = false;

canvas\_redraw();

}

}

if (IsDlgButtonChecked(hwnd, RadioButtonSP)) {

start.set\_center(POINT{ x, y });

canvas\_redraw();

return 0;

}

if (IsDlgButtonChecked(hwnd, RadioButtonEP)) {

end.set\_center(POINT{ x, y });

canvas\_redraw();

return 0;

}

if (IsDlgButtonChecked(hwnd, RadioButtonSetObs)) {

if (SetObsLButtonPressed) return 0;

ObsSet.push(new FilledRectangle(

hwnd, POINT{ x, y }, POINT{ x, y }, NULL, PumpkinColor)

);

//canvas\_redraw(); zero-size rect - redrow isn't needed

SetObsLButtonPressed = true;

return 0;

}

if (IsDlgButtonChecked(hwnd, RadioButtonSelectObs)) {

SelectedObs = ObsSetCollition(x,y);

if (!SelectedObs) {

return 0;

}

CapturesIsActive = true;

/\*POINT A = SelectedObs->get\_a();

POINT D = SelectedObs->get\_d();\*/

//char buff[100]("\0");

//std::string bb = std::to\_string(SelectedObs->get\_a().x);

//strcat\_s(buff, bb.c\_str());

//strcat\_s(buff, ";");

//bb = std::to\_string(SelectedObs->get\_a().y);

//strcat\_s(buff, bb.c\_str());

//MessageBoxA(hwnd, buff, "obs is finded", MB\_OK);

for (auto i = 0; i != 4; i++) {

auto A = SelectedObs->get\_i(i);

(CptrArr[i])->set\_center(A);

CptrRct[i].left = A.x - CaptureSize/2 + 1; CptrRct[i].top = A.y - CaptureSize / 2 + 1;

CptrRct[i].right = A.x + CaptureSize / 2 + 1; CptrRct[i].bottom = A.y + CaptureSize / 2 + 1;

}

canvas\_redraw();

}

}

return 0;

case WM\_LBUTTONUP:

{

SetObsLButtonPressed = false;

SelectedCptr = nullptr;

}

return 0;

default:

break;

}

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

}

return CallWindowProc(wpOrigCanvasProc, hWnd, uMsg, wParam, lParam);

}

class Rectangle\* ObsSetCollition(int x, int y)

{

if (ObsSet.is\_empty()) {

return nullptr;

}

for (auto obs = --ObsSet.end(); obs != ObsSet.begin(); --obs)

{

if (RectCollition(x, y, (\*obs)->get\_a(), (\*obs)->get\_c()))

return \*obs;

}

class Rectangle\* first\_obs = \*ObsSet.begin();

if (RectCollition(x, y, first\_obs->get\_a(), first\_obs->get\_c()))

return first\_obs;

return nullptr;

}

class Circle\* CptrSetColision(int x, int y)

{

// TODO: delete this statement

if (CptrArr.empty()) {

return nullptr;

}

for (auto p\_cptr = CptrArr.rbegin(); p\_cptr != CptrArr.rend(); p\_cptr++)

{

POINT left\_top = { (\*p\_cptr)->get\_center().x - CaptureSize, (\*p\_cptr)->get\_center().y - CaptureSize };

POINT right\_buttom = { (\*p\_cptr)->get\_center().x + CaptureSize, (\*p\_cptr)->get\_center().y - CaptureSize };

if (!RectCollition(x, y, left\_top, right\_buttom))

return nullptr;

return \*p\_cptr;

}

}

bool RectCollition(int x, int y, POINT A, POINT C)

{

if (((x > A.x && x < C.x) || (x > C.x && x < A.x)) &&

((y > A.y && y < C.y) || (y > C.y && y < A.y)))

return true;

return false;

}

void MessageBoxError(const std::exception& ex)

{

char buff[200]("\0");

strcat\_s(buff, ex.what());

strcat\_s(buff, "\n");

strcat\_s(buff, typeid(ex).name());

MessageBoxA(hwnd, buff, NULL, MB\_ICONERROR);

}

//void CreateWindowSeparator(int y, int CWidth) {

// CreateWindowA(

// "static", NULL, WS\_CHILD | WS\_VISIBLE | SS\_GRAYRECT,

// 3 \* CWidth / 4 + WSeparatorWidth, y,

// CWidth / 4 - WSeparatorWidth, WSeparatorWidth,

// hwnd, nullptr, nullptr, nullptr

// );

//};

void canvas\_redraw() {

HDC memDC = CreateCompatibleDC(cnvsDC);

HBITMAP memBM = CreateCompatibleBitmap(cnvsDC, cnvsRCT.right - cnvsRCT.left, cnvsRCT.bottom - cnvsRCT.top);

SelectObject(memDC, memBM);

HBRUSH OldBrush = SelectBrush(memDC, GetStockBrush(WHITE\_BRUSH));

Rectangle(memDC, cnvsRCT.left, cnvsRCT.top, cnvsRCT.right, cnvsRCT.bottom);

SelectBrush(memDC, OldBrush);

DeleteBrush(OldBrush);

try

{

start.draw(memDC);

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

return;

}

try

{

end.draw(memDC);

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

return;

}

try

{

for (auto obs = ObsSet.begin(); obs != ObsSet.end(); obs++) {

(\*obs)->draw(memDC);

}

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

return;

}

if (CapturesIsActive)

try

{

for (auto p\_cptr = CptrArr.begin(); p\_cptr != CptrArr.end(); p\_cptr++) {

(\*p\_cptr)->draw(memDC);

}

}

catch (const std::exception& ex)

{

MessageBoxError(ex);

return;

}

BitBlt(cnvsDC, 0, 0, cnvsRCT.right - cnvsRCT.left, cnvsRCT.bottom - cnvsRCT.top, memDC, 0, 0, SRCCOPY);

DeleteDC(memDC);

DeleteObject(memBM);

}

void SetGoalPoint(UINT uGOAL)

{

BOOL xSuccess, ySuccess;

long x, y;

switch (uGOAL)

{

case SetStartPoint:

x = GetDlgItemInt(hwnd, StartPointXEdit, &xSuccess, false);

y = GetDlgItemInt(hwnd, StartPointYEdit, &ySuccess, false);

if (xSuccess && ySuccess) {

start.set\_center(POINT{ x, y });

canvas\_redraw();

}

return;

case SetEndPoint:

x = GetDlgItemInt(hwnd, EndPointXEdit, &xSuccess, false);

y = GetDlgItemInt(hwnd, EndPointYEdit, &ySuccess, false);

if (xSuccess && ySuccess) {

end.set\_center(POINT{ x, y });

canvas\_redraw();

}

return;

default:

return;

}

}