// EyeDrawButton.cpp : implementation file
// This file contains the implementation of EyeDrawButton
// which is a base class for:
// ED_ShapeButton
// ED_ColorButton
// ED_UtilityButton
// The EyeDraw draw-with-the-eyes computer program
// 2005
// By Anna Cavender and Rob Hoselton
// Project Lead: Anthony Hornof
// https://www.cs.uoregon.edu/research/cm-hci/EyeDraw/

#pragma once
#include "stdafx.h"
#include "EyeDrawButton.h"
#include "EyeDrawDlg.h"
#include <windows.h>
#include <mmsystem.h>
#include ".\eyedrawbutton.h"

IMPLEMENT_DYNAMIC(EyeDrawButton, CButton)

// Standard Constructor
EyeDrawButton::EyeDrawButton()
{
    gazeCount = 0; /* number of times eye gaze samples on the button */
    isDown = false; /* starts in the up state */
    once = true; /* used for things that should only happen once */
}

// Destructor
EyeDrawButton::~EyeDrawButton(){}

BEGIN_MESSAGE_MAP(EyeDrawButton, CButton)
    // Message about eye movement from the Eyegaze thread (through EyeDrawDlg)
    ON_MESSAGE(UWM_ED_MOVE, OnEyeMovement)
    // Redraw message, maps to OnPaint
    ON_WM_PAINT()
    // Message from EyeDrawDlg about a change in program state
    ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
    // Message requesting a new bitmap image be displayed on the button
    ON_MESSAGE(UWM_ED_BITMAP_CHANGE, OnBitmapChange)
    ON_WM_WINDOWPOSCHANGED()
    // Timer message, maps to OnTimer
    ON_WM_TIMER()
    ON_WM_MOUSEMOVE()
    ON_WM_KEYUP()
    ON_MESSAGE(WM_KEYUP, OnKeyUp)
END_MESSAGE_MAP()

OnEyeMovement handles an eye movement message. The message will originate in the
GazeDataGetterThread in eyedata.h and pass through EyeDrawDlg where all buttons are
notified.

wParam is the x-coord of the eye location.
lParam is the y-coord of the eye location.

1. Determine if gaze location is on the button
   (if not, reset the gazeCount and make sure button is "up").
2. Increment gazeCount.
3. If there have been enough gazeCounts (the user has been looking at the button long enough, click yourself.

There is slight pause between the "down" and "up" position, otherwise the change would not be visible.

LRESULT EyeDrawButton::OnEyeMovement(WPARAM wParam, LPARAM lParam) {

// Construct a CPoint object to pass to WindowFromPoint
CPoint point(LOWORD(lParam), HIWORD(lParam));
CWnd *windowUnderEye = WindowFromPoint(point);
CRect rt;
this->GetWindowRect(&rt);

// Extract the x- y- coordinates from lParam
int eye_x = point.x - rt.left;
int eye_y = point.y - rt.top;

// if the gaze point is within the button boundary
if (windowUnderEye == this) {

// Only manipulate the eyecursor if it has been initialized (once is false)
// and if the "Dot" button hasn't been clicked (clutch is false).
if (!once && !clutch) {
    // Erase the old eyecursor position
    eyeCursor->erase(&buttontempDC);
    InvalidateRect(eyeCursor->getRect(), FALSE);

    // Move cursor to new location
    eyeCursor->move(eye_x, eye_y);

    // Draw the new eyecursor
    eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
    InvalidateRect(eyeCursor->getRect(), FALSE);
}

/* We count the number of gaze points at this button. We cannot rely on a fixation
message from detectFixation because we want the button to be clicked by looking
anywhere inside the button. gazeCount keeps track of the number of consecutive
samples within the button boundary. */
gazeCount++;

// After 30 samples press the button. Allow for 5 samples before releasing the button
// for a better visualization of a button click
if (gazeCount == 30) {
    isDown = true;
    redraw();
}

// After 35 gaze samples consider the button clicked and notify the main app of the
// new state
if (gazeCount == 35) {
    isDown = false;
    // Do appropriate action for button click
    clicked();
    redraw();
}

// Reset gazeCount so the button can be clicked again
if (gazeCount == 75) {
    gazeCount = 0;
}
} else {
    // Set gazeCount back to 0 for this button
    gazeCount = 0;
    if (isDown) {
        isDown = false;
        redraw();
    }
    if (!once && !clutch) {
// Erase the old eyecursor position
eyeCursor->erase(&buttontempDC);
InvalidateRect(eyeCursor->getRect(),FALSE);
}

return 0;

/******************************************************************************
  This function allows each type of button class to respond differently to a "click".  
  It must be defined in all derived classes.
  1. Play the CLICK sound.
  2. Post a message to the parent notifying it of the click.
*******************************************************************************/

void EyeDrawButton::clicked()
{
  // after enough eye-samples, button gets clicked
  // play clicking sound
  if (Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
  // post message to parent window notifying it of a click
  CWnd *parent = GetParent();
  parent->PostMessage(UWM_ED_STATE_CHANGE,value,type);
}

/******************************************************************************
  The redraw function redraws the button based on "up" or "down" states. Derived classes
  can redefine this for a different appearance.
  1. Determine if button is "up" or "down" and draw sunken or raised.
  2. Draw the bitmap if this button has one.
*******************************************************************************/

void EyeDrawButton::redraw()
{
  pMemDC = new CDC;
  pMemDC->CreateCompatibleDC(&buttontempDC);

  //Get the rectangle area of this button
  this->GetClientRect(&button_rect);

  //redraw grey square either sunken or raised
  if ( (isDown) ) {
    buttontempDC.FillSolidRect(button_rect, RGB_GREY);
    buttontempDC.DrawEdge(button_rect,EDGE_SUNKEN,BF_RECT);
  } else {
    buttontempDC.FillSolidRect(button_rect, RGB_GREY);//grey
    buttontempDC.DrawEdge(button_rect,EDGE_RAISED,BF_RECT);
  }

  // If there is a bitmap to draw on the button then do it
  if (hasBitmap) {
    CBitmap *oldBitmap = pMemDC->SelectObject(&bitmap);
    buttontempDC.StretchedBlt(button_rect.Width()/10,button_rect.Height()/10,button_rect.Width()*4/5,button_rect.Height()*4/5,
                      pMemDC, 0, 0,100, 85,
                      SRCCOPY);
    pMemDC->SelectObject(oldBitmap);
  }
}

void EyeDrawButton::OnPaint()
{
  // device context for painting
  CPaintDC paintdc(this);
this->GetClientRect(&button_rect);

// The very first time OnPaint is called, initialize the device contexts
// and initialize the eye cursor, then draw everything needed.
if (once){
    // Initialize mydc
    mydc = GetDC();

    // Initialize the dc for temporary background
    buttontempDC.CreateCompatibleDC(mydc);
    image.CreateCompatibleBitmap(mydc, button_rect.Width(), button_rect.Height());
    buttontempDC.SelectObject(&image);
    buttontempDC.FillSolidRect(&button_rect, RGB_GREY); // background
    image.DeleteObject();

    // Initialize the dc for permanent background
    buttonmemoryDC.CreateCompatibleDC(mydc);
    memoryRectBg.CreateCompatibleBitmap(mydc, button_rect.Width(), button_rect.Height());
    buttonmemoryDC.SelectObject(&memoryRectBg);
    buttonmemoryDC.FillSolidRect(&button_rect, RGB_GREY); // background

    redraw();

    // Initialize the eye cursor
    eyeCursor = new EyeCursor(mydc,button_rect, RGB_GREY);
    eyeCursor->setColor(RGB_BLUE);
    eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
    eyeCursor->erase(&buttontempDC);
    InvalidateRect(eyeCursor->getRect(), FALSE);

    // Set the timer to turn off the mouse after 7 seconds
    this->SetTimer(4,7000,0);
    once = false;
}

// Blt the temporary device context (because it contains everything) to screen
paintdc.BitBlt( button_rect.left,
    button_rect.top,
    button_rect.Width(),
    button_rect.Height(),
    &buttontempDC,
    0,
    0,
    SRCCOPY);

LRESULT EyeDrawButton::OnStateChange(WPARAM wParam, LPARAM lParam){
    //if the program’s new state is the same as the button’s type,
    //then this tool has been selected.
    return 0;
}

LRESULT EyeDrawButton::OnBitmapChange(WPARAM wParam, LPARAM lParam){
    //if the program’s new state is the same as the button’s type,
    //then this tool has been selected.
    return 0;
}
1. Convert the filename into a LPTSTR.
2. Delete the old bitmap.
3. Load the new one.
4. Redraw the button.

```cpp
LRESULT EyeDrawButton::OnBitmapChange(WPARAM wParam, LPARAM lParam)
{
    // get the filename of the bitmap
    filename = (CString *)lParam;
    // These next two lines convert a filename into a LPTSTR so that it can be passed to
    // LoadBMPImage
    LPTSTR lpsz = new TCHAR[filename->GetLength()+1];
    _tcscpy(lpsz, *filename);
    if(bitmap.m_hObject != NULL){
        bitmap.DeleteObject();
    }
    // Load the bitmap from the file
    hasBitmap = LoadBMPImage(lpsz, bitmap, NULL);
    // redraw the button
    redraw();
    delete lpsz;
    return 0;
}
```

This function gets called when mouse is moved. Usually the mouse is not visible,
so this function makes it visible for a short time because it was moved.

```cpp
void EyeDrawButton::OnMouseMove(UINT nFlags, CPoint point)
{
    ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
    this->SetTimer(4, 7000, 0);
    CButton::OnMouseMove(nFlags, point);
}
```

This function gets called when timer started by a mouse move runs out. This means
that the mouse has been stationary for long enough to make it disappear again.

```cpp
void EyeDrawButton::OnTimer(UINT nIDEvent)
{
    ::SetCursor(AfxGetApp()->LoadStandardCursor(0));
    CButton::OnTimer(nIDEvent);
}
```

This function gets called when key from the keyboard is pressed. It is used to
simulate some of the appropriate actions of the F keys.

```cpp
void EyeDrawButton::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    if( nChar == VK_F1 || nChar == VK_F12 ){
        CWnd *parent = GetParent();
        parent->PostMessage(WM_KEYUP,nChar,nRepCnt);
    }
    CButton::OnKeyUp(nChar, nRepCnt, nFlags);
}
```
LRESULT EyeDrawButton::OnKeyUp( WPARAM wParam, LPARAM lParam )
{
    if( wParam == VK_F1 || wParam == VK_F12 ){
        CWnd *parent = GetParent();
        parent->PostMessage(WM_KEYUP,wParam,lParam);
    }
    return 0;
}

IMPLEMENT_DYNAMIC(ED_ShapeButton, EyeDrawButton)

// Standard Constructor
// - defaults are the button is not selected and the shape is not filled
ED_ShapeButton::ED_ShapeButton()
{
    isToolSelected = false;
currentFill = false;
}

// Constructor taking
ED_ShapeButton::ED_ShapeButton(int value)
{
    type = SHAPE_STATE;    /* indicates which class it belongs to (SHAPE_STATE here)*/
    value = value;          /* indicates what it does if clicked */

    // initially not selected
    isToolSelected = false;
    // initially not filled
    currentFill = false;
}

// Destructor
ED_ShapeButton::~ED_ShapeButton(void){}

BEGIN_MESSAGE_MAP(ED_ShapeButton, EyeDrawButton)
    // Message notifying button of change in program state
    ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
    // ON_WM_KEYUP()
END_MESSAGE_MAP()

/****************************************************************************************
 This function allows shape buttons to respond differently to a "click". It is called
 from within the base class's OnEyeMovement.

 1. Play the CLICK sound.
 2. Post a message to the parent notifying it of the click.
 3. If this button is selected, toggle whether the shape is filled.
 4. Notify the parent of the current fill status.
****************************************************************************************/
void ED_ShapeButton::clicked()
{
    // after enough eye-samples, button gets clicked
    // play clicking sound
    if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
    // post message to parent that button was clicked
    CWnd *parent = GetParent();
    // type = SHAPE_STATE, and value = what shape this button represents
    parent->PostMessage(UWM_ED_STATE_CHANGE,value,type);

    if(isToolSelected) {
        // change the current fill after first click
        currentFill = !currentFill;
    }
    // and tell the parent the fill has changed
    parent->PostMessage(UWM_ED_STATE_CHANGE, currentFill, FILL);
}
/*******************************************************************************************/
void ED_ShapeButton::redraw()
{
  // call the base class's redraw
  EyeDrawButton::redraw();

  // draw the appropriate shape with the appropriate color (grey if not selected) and fill
  CPen pen(PS_SOLID, 5, isToolSelected ? currentColor : RGB_DARKGREY);
  CPen* oldPen = buttontempDC.SelectObject(&pen);
  COLORREF fill = (currentFill ? (isToolSelected ? currentColor : RGB_DARKGREY) :
    RGB_GREY);
  CBrush brush(fill);
  CBrush* oldBrush = (CBrush*)buttontempDC.SelectObject(&brush);

  switch (value){
  case FREE:
    buttontempDC.MoveTo(10,10);
    buttontempDC.LineTo(20,20);
    buttontempDC.LineTo(15,25);
    buttontempDC.LineTo(button_rect.Width()/2,button_rect.Height()/2);
    buttontempDC.LineTo(button_rect.Width()/2 + 5, button_rect.Height()/2 + 10);
    buttontempDC.LineTo(button_rect.Width()/2 + 10, button_rect.Height()/2 + 15);
    buttontempDC.LineTo(button_rect.Width() - 15, button_rect.Height()/2 - 20);
    buttontempDC.LineTo(button_rect.Width() - 10, button_rect.Height() - 10);
    break;
  case LINE:
    if (currentFill){
      buttontempDC.MoveTo(40,20);
      buttontempDC.LineTo(20,button_rect.Height()-20);
      buttontempDC.LineTo(button_rect.Width()-20,button_rect.Height()-20);
      buttontempDC.LineTo(button_rect.Width() - 20, button_rect.Height() / 2 - 20);
    }else{
      buttontempDC.MoveTo(20,20);
      buttontempDC.LineTo(button_rect.Width() - 20, button_rect.Height() - 20);
    }
    break;
  case CIRCLE:
    if (isToolSelected ){
      buttontempDC.Ellipse(10,20,button_rect.Width()-10,button_rect.Height()-20);
    }else{
      buttontempDC.Ellipse(10,20,button_rect.Width()-10,button_rect.Height()-20);
    }
    break;
  case RECTANGLE:
    if (isToolSelected ){
      buttontempDC.Rectangle(20,20,button_rect.Width()-20,button_rect.Height()-20);
    }else{
      buttontempDC.Rectangle(20,20,button_rect.Width()-20,button_rect.Height()-20);
    }
    break;
  }
  buttontempDC.SelectObject(oldPen);
  buttontempDC.SelectObject(oldBrush);

  // And copy the tempDC to the memoryDC
  buttonmemoryDC.BitBlt( button_rect.left,
    button_rect.top,
    button_rect.Width(),
    button_rect.Height(),
    buttontempDC,
)
if(!once){
    // Draw the new eyeCursor
    eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
    InvalidateRect(eyeCursor->getRect(),FALSE);
    // Erase the old eyecursor position
    eyeCursor->erase(&buttontempDC);
    InvalidateRect(eyeCursor->getRect(),FALSE);
}
InvalidateRect(button_rect,FALSE);
// Erase the old eyecursor position
eyeCursor->erase(&buttontempDC);
InvalidateRect(eyeCursor->getRect(),FALSE);
}
InvalidateRect(button_rect,FALSE);

OnStateChange handles messages from EyeDrawDlg about changes in button state.

lParam represent the new state type
wParam represent the new value or change

Shape buttons only care about the following conditions:
1. If the new state type is SHAPE_STATE and the value is this button's value, this
   button is now selected. If its not this button's value, it is deselected.
2. If the new state type is a change in color, change this button's color to reflect
   that.
3. If the new state type is a stamp, this button can't possibly be selected.

*****************************************************************************************/
LRESULT ED_ShapeButton::OnStateChange(WPARAM wParam, LPARAM lParam)
{
    //if the program's new state is the same as the button's type,
    //then this tool has been selected.
    if((int)lParam == SHAPE_STATE){
        if (value == (int)wParam){
            if(!isToolSelected){
                isToolSelected = true;
                redraw();
            }
        }else{
            if(isToolSelected){
                isToolSelected = false;
                redraw();
            }
        }
    }
    // if the message is of a new color, reset the button'e shape color
    if( (int)lParam == COLOR ){
        currentColor = (COLORREF)wParam;
        redraw();
    }
    if( (int)lParam == STAMP_STATE ){
        if(isToolSelected){
            isToolSelected = false;
            redraw();
        }
    }
    return 0;
}

ULONG ED_ShapeButton::GetColor(){
    return currentColor;
}

IMPLEMENT_DYNAMIC(ED_ShapeButton, EyeDrawButton)

// Standard Constructor sets the button's color to white and not selected.
ED_ColorButton::ED_ColorButton()
{
    color = RGB(255, 255, 255);
    isColorSelected = false;
}

// Constructor taking a value and a color.
ED_ColorButton::ED_ColorButton(int value1, COLORREF color1)
{
    type = COLOR;  /* indicates the class it belongs to, determined by color*/
    value = value1;  /* indicates what it does if clicked */
    color = color1;
    // The black color button is the initial program default
    if (color == RGB_BLACK) {
        isColorSelected = true;
    } else {
        isColorSelected = false;
    }
}

// Destructor
ED_ColorButton::~ED_ColorButton(void){}

BEGIN_MESSAGE_MAP(ED_ColorButton, EyeDrawButton)
    // Message notifying button of change in program state
    ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
END_MESSAGE_MAP()

/******************************************
This function allows color buttons to respond differently to a "click". It is called
from within the base class's OnEyeMovement.

1. Play the CLICK sound.
2. Post a message to the parent notifying it of the click.
*******************************************/
void ED_ColorButton::clicked()
{
    // play clicking sound
    if(Sound) PlaySound(\"EyeDraw/Sounds/CLICK.WAV\", NULL, SND_FILENAME | SND_ASYNC);
    // post message to parent that button was clicked
    CWnd *parent = GetParent();
    // type = COLOR_STATE, and value = what color this button represents
    parent->PostMessage(UWM_ED_STATE_CHANGE, color, type);
}

/******************************************
The redraw function redraws the button as needed.
1. Call the base class's DrawItem to get the standard appearance.
2. Draw a rectangle that is small if not selected and larger if the button is selected.
3. Resave the new button area into the location retrieved by the EyeCursor after it
   moves. Otherwise, the EyeCursor will paste back the wrong image.
*******************************************/
void ED_ColorButton::redraw()
{
    // Call the base class's DrawItem
    EyeDrawButton::redraw();

    // Draw the rectangle with this button's color, either small or large
    if (isColorSelected) {
        buttontempDC.FillSolidRect(button_rect.Width()/8, button_rect.Height()/8, button_rect.Width() * 3/4, button_rect.Height() * 3/4, color);
    } else {
        buttontempDC.FillSolidRect(button_rect.Width()/3, button_rect.Height()/3, button_rect.Width()/3, button_rect.Height()/3, color);
    }

    // And copy the tempDC to the memoryDC
    buttonmemoryDC.BitBlt( button_rect.left,
if(!once){
    // Draw the new eyeCursor
    eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
    InvalidateRect(eyeCursor->getRect(), FALSE);
    // Erase the old eyecursor position
    eyeCursor->erase(&buttontempDC);
    InvalidateRect(eyeCursor->getRect(), FALSE);
}
InvalidateRect(button_rect, FALSE);

/***********************************************************/
OnStateChange handles messages from EyeDrawDlg about changes in button state.

lParam represent the new state type
wParam represent the new value or change

Color buttons only care about the following conditions:
1. If the new state type is COLOR and the color is this button's color, this
   button is now selected. If its not this button’s color, it is deselected.
/**************************************************************/
LRESULT ED_ColorButton::OnStateChange(WPARAM wParam, LPARAM lParam){

    // if the program's new state is the same as the button's type,
    // then this tool has been selected.
    if( ((int)lParam == COLOR )){
        if( color == (COLORREF)wParam ) {
            if(!isColorSelected){
                isColorSelected = true;
                redraw();
            }
        }else{
            if(isColorSelected){
                isColorSelected = false;
                redraw();
            }
        }
    }
    return 0;
}

IMPLEMENT_DYNAMIC(ED_UtilityButton, EyeDrawButton)

// Standard Constructor with generic defaults
ED_UtilityButton::ED_UtilityButton()
{
    gazeCount = 0; /* number of times eye gaze samples on the button */
    isDown = false; /* starts in the up state */
    name = ""; /* default text is nothing */
}

// Constructor taking a state and two texts for when button is selected and deselected.
ED_UtilityButton::ED_UtilityButton(int state, CString selectedName1, CString
deselectedName1, COLORREF fontcolor)
gazeCount = 0; /* number of times eye gaze samples on the button */

isDown = false; /* starts in the up state */
isToolSelected = false; /* starts out deselected */
hasBitmap = false; /* originally no bitmaps are displayed on button */
type = state; /* the type of button */
value = 0;

selectedIndex = selectedName1; /* the text to display when the button is selected */
deselectedName = deselectedName1; /* the text to display when the button is deselected*/
name = deselectedName1; /* show the deselected name first */
color = fontcolor; /* font color for this button */
clutch = false; /* the "Dot" button is not yet selected */

// Destructor
ED_UtilityButton::~ED_UtilityButton(){

BEGIN_MESSAGE_MAP(ED_UtilityButton, EyeDrawButton)

    // Message notifying button of change in program state
    ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)

    // Message requesting the button change the text written on it.
    ON_MESSAGE(UWM_ED_CAPTION_CHANGE, OnCaptionChange)

END_MESSAGE_MAP()

/* This function allows utility buttons to respond differently to a "click". It is called
from within the base class's OnEyeMovement.

1. Play the CLICK sound.
2. Post a message to the parent notifying it of the click.
*/
void ED_UtilityButton::clicked()
{
    // after enough eye-samples, button gets clicked
    // play clicking sound
    if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);

    // post message to parent that button was clicked
    CWnd *parent = GetParent();
    // type = the state change that this button represents, and value is not being used in
    // this case
    parent->PostMessage(UWM_ED_STATE_CHANGE,value,type);
}

/* The redraw function redraws the button as needed.

1. Call the base class's redraw to get the standard appearance.
2. Use built in fonts and sizes to draw the name on the button. The name will already
have the appropriate text.
3. Resave the new button area into the location retrieved by the EyeCursor after it
moves. Otherwise, the EyeCursor will paste back the wrong image.
*/
void ED_UtilityButton::redraw()
{
    // call the base class's DrawItem
    EyeDrawButton::redraw();

    this->GetClientRect(&button_rect);

    // if this is a grid button, and it is selected, draw the grid on it
    CPen pen(PS_SOLID,1,RGB_BLACK);
    CPen* oldPen = buttontempDC.SelectObject(&pen);
    if(type == GRID_STATE){
        for (int i = button_rect.Width()/8; i <= 7*button_rect.Width()/8;
            i+=button_rect.Width()/8){
            for (int j = button_rect.Height()/8; j < 7*button_rect.Height()/8;
                j+=button_rect.Height()/8){
                buttontempDC.Ellipse(i,j,i+2,j+2);
            }
        }
    }
}
buttontempDC.SelectObject(oldPen);
// create a font and draw the button's text (name)
CFont font;
font.CreateFont(button_rect.Height()/3, button_rect.Width()/10, 0, 0, FW_BOLD, 0, 0, 0, 0, 0, "Arial");
CFont* def_font = buttontempDC.SelectObject(&font);
buttontempDC.SetBkColor(RGB_GREY);
buttontempDC.SetTextColor(color);
buttontempDC.DrawText(name, button_rect, DT_CENTER|DT_VCENTER|DT_SINGLELINE);
buttontempDC.SelectObject(def_font);
font.DeleteObject();

// And copy the tempDC to the memoryDC
buttonmemoryDC.BitBlt(button_rect.left, button_rect.top, button_rect.Width(), button_rect.Height(), &buttontempDC, 0, 0, SRCCOPY);

if(!once){
    // Draw the new eyeCursor
    eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
    InvalidateRect(eyeCursor->getRect(), FALSE);
    // Erase the old eyecursor position
    eyeCursor->erase(&buttontempDC);
    InvalidateRect(eyeCursor->getRect(), FALSE);
    InvalidateRect(button_rect, FALSE);
}
InvalidateRect(button_rect, FALSE);

if(clutch){
    // If clutch is true, this means that the only button receiving eye movement data
    // is the "Dot On"/"Dot Off" button, so
    // Park the eyeCursor
    eyeCursor->move(button_rect.left + button_rect.Width()/2 - 5, button_rect.top + button_rect.Height()/3 - 5);
    // Draw the new eyeCursor
    eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
    InvalidateRect(eyeCursor->getRect(), FALSE);
}

OnCaptionChange handles messages requesting a change in the text written on the button. wParam and lParam are not being used here.
1. Toggle the selectedName and deselectedName.
2. Redraw the button.
LRESULT ED_UtilityButton::OnCaptionChange(WPARAM wParam, LPARAM lParam){
    // changes the caption of the button when clicked
    if(isToolSelected){
        name.SetString(deselectedName);
        isToolSelected = false;
        clutch = false;
    }else{ //isToolSelected == false
        name.SetString(selectedName);
        isToolSelected = true;
        if(type == DOT_STATE){
            name.SetString(deselectedName);
            isToolSelected = false;
            clutch = false;
        }
    }
}
```cpp
clutch = true;
}
redraw();
return 0;

IMPLEMENT_DYNAMIC(ED_StampButton, EyeDrawButton)

// Standard constructor with default values
ED_StampButton::ED_StampButton()
{
    gazeCount = 0; /* number of times eye gaze samples on the button */
    isDown = false; /* button starts in "up" position */
    hasBitmap = false; /* stamp buttons have resource ids, not bitmaps */
}

// Constructor taking an id for the resource id of the button's bitmap
ED_StampButton::ED_StampButton(int id){
    gazeCount = 0; /* number of times eye gaze samples on the button */
    isDown = false; /* starts in the up state */
    type = STAMP; /* the type of button */
    hasBitmap = false; /* stamp buttons have resource ids, not bitmaps */
    resourceID = id; /* resource id of this button's bitmap */
}

// Destructor
ED_StampButton::~ED_StampButton(){

BEGIN_MESSAGE_MAP(ED_StampButton, EyeDrawButton)
    // Message notifying button of change in program state
    ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
END_MESSAGE_MAP()

// This function allows stamp buttons to respond differently to a "click". It is called
// from within the base class's OnEyeMovement.
void ED_StampButton::clicked()
{
    // after enough eye-samples, button gets clicked
    // play clicking sound
    if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
    // post message to parent that button was clicked
    CWnd *parent = GetParent();
    // type = the state change that this button represents, and value is not being used in
    // this case
    parent->PostMessage(UWM_ED_STATE_CHANGE, resourceID, type);
}

// The redraw function redraws the button as needed.
void ED_StampButton::redraw()
{
    EyeDrawButton::redraw();
    CBitmap bmp;
```
bmp.LoadBitmap(resourceID);
HBITMAP hBitmap = (HBITMAP) bmp.GetSafeHandle();

// use a temporary DC just so that we can stretch blt it to the real DC
CDC *tmpDC = this->GetDC();
tmpDC->FillSolidRect(0, 0, 70, 70, RGB_GREY);

DrawTransparentBitmap(*tmpDC, // The destination DC.
    hBitmap, // The bitmap to be drawn.
    0, // X coordinate.
    0, // Y coordinate.
    0x00FFFFFF); // The color for transparent
    // pixels (white, in this example).

// Then stretch it to fit the button
buttontempDC.StretchBlt(button_rect.Width() / 8,
    button_rect.Height() / 8,
    3 * button_rect.Width() / 4,
    3 * button_rect.Height() / 4,
    tmpDC, 0, 0, 70, 70, SRCCOPY);

if(isToolSelected){
    //draw a green box around it
    CPen pen(PS_SOLID, 4, RGB_GREEN);

    // Select the pen into the device context.
    CPen* pOldPen = (CPen*) buttontempDC.SelectObject(&pen);

    buttontempDC.MoveTo(button_rect.left + 10, button_rect.top + 10);
    buttontempDC.LineTo(button_rect.right - 10, button_rect.top + 10);
    buttontempDC.LineTo(button_rect.right - 10, button_rect.bottom - 10);
    buttontempDC.LineTo(button_rect.left + 10, button_rect.bottom - 10);
    buttontempDC.LineTo(button_rect.left + 10, button_rect.top + 10);

    //Restore original device context object
    buttontempDC.SelectObject(pOldPen);
}

// And copy the tempDC to the memoryDC
buttonmemoryDC.BitBlt(button_rect.left,
    button_rect.top,
    button_rect.Width(),
    button_rect.Height(),
    &buttontempDC,
    0,
    0,
    SRCCOPY);

if(!once){
    // Draw the new eyecursor
    eyecursor->draw(&buttontempDC, &buttonmemoryDC);
    InvalidateRect(eyecursor->getRect(), FALSE);
    // Erase the old eyecursor position
    eyecursor->erase(&buttontempDC);
    InvalidateRect(eyecursor->getRect(), FALSE);

    InvalidateRect(button_rect, FALSE);
}

InvalidateRect(button_rect, FALSE);

OnStateChange handles messages from EyeDrawDlg about changes in button state.

lParam represent the new state type
wParam represent the new value or change
Stamp buttons only care about the following conditions:
1. If the new state type is STAMP_STATE and the value is this button's value, this
   button is now selected. If its not this button's value, it is deselected.

LRESULT ED_StampButton::OnStateChange(WPARAM wParam, LPARAM lParam)
{
    //if the program's new state is the same as the button's type,
    //then this tool has been selected.
    if((int)lParam == STAMP){
        if (resourceID == (int)wParam){
            isToolSelected = true;
            redraw();
        }
    } else {  
        if(isToolSelected){
            isToolSelected = false;
            redraw();
        }
    }

    return 0;
}

Set proper mapping mode.
SetMapMode(hdcTemp, GetMapMode(hdc));

Save the bitmap sent here, because it will be overwritten.
BitBlt(hdcSave, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0, SRCCOPY);

Set the background color of the source DC to the color.
// contained in the parts of the bitmap that should be transparent
cColor = SetBkColor(hdcTemp, cTransparentColor);

Create the object mask for the bitmap by performing a BitBlt
from the source bitmap to a monochrome bitmap.
BitBlt(hdcObject, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0, SRCCOPY);

Set the background color of the source DC back to the original
// color.
SetBkColor(hdcTemp, cColor);

Create the inverse of the object mask.
BitBlt(hdcBack, 0, 0, ptSize.x, ptSize.y, hdcObject, 0, 0, NOTSRCCOPY);

Copy the background of the main DC to the destination.
BitBlt(hdcMem, 0, 0, ptSize.x, ptSize.y, hdc, xStart, yStart, SRCCOPY);

Mask out the places where the bitmap will be placed.
BitBlt(hdcMem, 0, 0, ptSize.x, ptSize.y, hdcObject, 0, 0, SRCAND);

Mask out the transparent colored pixels on the bitmap.
BitBlt(hdcTemp, 0, 0, ptSize.x, ptSize.y, hdcBack, 0, 0, SRCAND);

XOR the bitmap with the background on the destination DC.
BitBlt(hdcMem, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0, SRCPAINT);

Copy the destination to the screen.
BitBlt(hdc, xStart, yStart, ptSize.x, ptSize.y, hdcMem, 0, 0, SRCCOPY);

Place the original bitmap back into the bitmap sent here.
BitBlt(hdcTemp, 0, 0, ptSize.x, ptSize.y, hdcSave, 0, 0, SRCCOPY);

Delete the memory bitmaps.
DeleteObject(SelectObject(hdcBack, bmBackOld));
DeleteObject(SelectObject(hdcObject, bmObjectOld));
DeleteObject(SelectObject(hdcMem, bmMemOld));
DeleteObject(SelectObject(hdcSave, bmSaveOld));

Delete the memory DCs.
DeleteDC(hdcMem);
DeleteDC(hdcBack);
DeleteDC(hdcObject);
DeleteDC(hdcSave);
DeleteDC(hdcTemp);

IMPLEMENT_DYNAMIC(ED_KeyButton, EyeDrawButton)

// Standard Constructor with generic defaults
ED_KeyButton::ED_KeyButton()
{
    gazeCount = 0;  /* number of times eye gaze samples on the button */
    isDown = false; /* starts in the up state */
    value = "";    /* default text is nothing */
}
// Constructor taking a state and two texts for when button is selected and deselected.
ED_KeyButton::ED_KeyButton(CString value1)
{
    gazeCount = 0;   /* number of times eye gaze samples on the button */
    isDown = false;  /* starts in the up state */
    hasBitmap = false;  /* originally no bitmaps are displayed on button */
    type = KEY;       /* the type of button */
    value = value1;   /* the value of button */
    clutch = false;   /* the "Dot" button is not yet selected */
}

// Destructor
ED_KeyButton::~ED_KeyButton()
{}

BEGIN_MESSAGE_MAP(ED_KeyButton, EyeDrawButton)
END_MESSAGE_MAP()

/****************************************************************************************
This function allows key buttons to respond differently to a "click". It is called from
within the base class's OnEyeMovement.

1. Play the CLICK sound.
2. Post a message to the parent notifying it of the click.
****************************************************************************************/

void ED_KeyButton::clicked()
{
    // after enough eye-samples, button gets clicked
    // play clicking sound
    if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
    // post message to parent that button was clicked
    CWnd *parent = GetParent();
    // type = the state change that this button represents, and value is not being used in
    // this case
    parent->PostMessage(UWM_ED_STATE_CHANGE,(WPARAM)&value,type);
}

/****************************************************************************************
The redraw function redraws the button as needed.

1. Call the base class's redraw to get the standard appearance.
2. Use built in fonts and sizes to draw the name on the button.
3. Resave the new button area into the location retrieved by the EyeCursor after it moves.
   Otherwise, the EyeCursor will paste back the wrong image.
****************************************************************************************/

void ED_KeyButton::redraw()
{
    // call the base class's DrawItem
    EyeDrawButton::redraw();
    this->GetClientRect(&button_rect);

    if(isDown) buttontempDC.FillSolidRect(button_rect, RGB_GREEN);
    // create a font and draw the button's text (name)
    CFont font;
    font.CreateFont(button_rect.Height()/3,10,0,0,FW_BOLD,0,0,0,0,0,"Arial");
    CFont* def_font = buttontempDC.SelectObject(&font);
    buttontempDC.SetBkColor(isDown? RGB_GREEN : RGB_GREY);
    buttontempDC.DrawText(value, button_rect, DT_CENTER|DT_VCENTER|DT_SINGLELINE);
    buttontempDC.SelectObject(def_font);
    font.DeleteObject();

    // And copy the tempDC to the memoryDC
    buttonmemoryDC.BitBlt(button_rect.left,
                           button_rect.top,
                           button_rect.Width(),
                           button_rect.Height(),
                           buttontempDC, 0, 0, SRCCOPY);
}

if(!once){
    // Draw the new eyeCursor
    eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
    InvalidateRect(eyeCursor->getRect(),FALSE);
    // Erase the old eyecursor position
    eyeCursor->erase(&buttontempDC);
    InvalidateRect(eyeCursor->getRect(),FALSE);

    InvalidateRect(button_rect,FALSE);
}

InvalidateRect(button_rect,FALSE);

if(clutch){
    // If clutch is true, this means that the only button receiving eye movement data
    // is the "Dot On"/"Dot Off" button, so
    // Park the eyeCursor
    eyeCursor->move(button_rect.left + button_rect.Width()/2 - 5, button_rect.top +
                    button_rect.Height()/3 - 5);

    // Draw the new eyeCursor
    eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
    InvalidateRect(eyeCursor->getRect(),FALSE);
}