

```

1 // EyeDrawButton.cpp : implementation file
2 // This file contains the implementation of EyeDrawButton
3 // which is a base class for:
4 // ED_ShapeButton
5 // ED_ColorButton
6 // ED_UtilityButton
7
8 // The EyeDraw draw-with-the-eyes computer program
9 // 2005
10 // By Anna Cavender and Rob Hoselton
11 // Project Lead: Anthony Hornof
12 // https://www.cs.uoregon.edu/research/cm-hci/EyeDraw/
13
14 #pragma once
15 #include "stdafx.h"
16 #include "EyeDrawButton.h"
17 #include "EyeDrawDlg.h"
18 #include <windows.h>
19 #include <mmsystem.h>
20 #include ".\eyedrawbutton.h"
21
22 ///////////////////////////////////////////////////
23 // EyeDrawButton
24 ///////////////////////////////////////////////////
25
26 IMPLEMENT_DYNAMIC(EyeDrawButton, CButton)
27
28 //Standard Constructor
29 EyeDrawButton::EyeDrawButton()
30 {
31     gazeCount = 0;           /* number of times eye gaze samples on the button */
32     isDown = false;          /* starts in the up state */
33     once = true;             /* used for things that should only happen once */
34 }
35
36 // Destructor
37 EyeDrawButton::~EyeDrawButton(){}
38
39
40 BEGIN_MESSAGE_MAP(EyeDrawButton, CButton)
41     // Message about eye movement from the Eyegaze thread (through EyeDrawDlg)
42     ON_MESSAGE(UWM_ED_MOVE, OnEyeMovement)
43
44     // Redraw message, maps to OnPaint
45     ON_WM_PAINT()
46
47     // Message from EyeDrawDlg about a change in program state
48     ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
49
50     // Message requesting a new bitmap image be displayed on the button
51     ON_MESSAGE(UWM_ED_BITMAP_CHANGE, OnBitmapChange)
52     ON_WM_WINDOWPOSCHANGED()
53
54     // Timer message, maps to OnTimer
55     ON_WM_TIMER()
56     ON_WM_MOUSEMOVE()
57
58     ON_WM_KEYUP()
59     ON_MESSAGE(WM_KEYUP, OnKeyUp)
60 END_MESSAGE_MAP()
61
62 /*****
63     OnEyeMovement handles an eye movement message. The message will originate in the
64     GazeDataGetterThread in eyedata.h and pass through EyeDrawDlg where all buttons are
65     notified.
66
67     wParam is the x-coord of the eye location.
68     lParam is the y-coord of the eye location.
69
70     1. Determine if gaze location is on the button
71     (if not, reset the gazeCount and make sure button is "up").

```

```

72     2. Increment gazeCount.
73     3. If there have been enough gazeCounts (the user has been looking at the button
74         long enough, click yourself.
75
76     There is slight pause between the "down" and "up" position, otherwise the change
77     would not be visible.
78     *****/
79 LRESULT EyeDrawButton::OnEyeMovement(WPARAM wParam, LPARAM lParam){
80
81     // Construct a CPoint object to pass to WindowFromPoint
82     CPoint point(LOWORD(lParam),HIWORD(lParam));
83     CWnd *windowUnderEye = WindowFromPoint(point);
84     CRect rt;
85     this->GetWindowRect(&rt);
86
87     // Extract the x- y- coordinates from lParam
88     int eye_x = point.x - rt.left;
89     int eye_y = point.y - rt.top;
90
91     // if the gaze point is within the button boundary
92     if(windowUnderEye == this){
93
94         // Only manipulate the eyecursor if it has been initialized (once is false)
95         // and if the "Dot" button hasn't been clicked (clutch is false).
96         if(!once && !clutch){
97             // Erase the old eyecursor position
98             eyeCursor->erase(&buttontempDC);
99             InvalidateRect(eyeCursor->getRect(),FALSE);
100
101             // Move cursor to new location
102             eyeCursor->move(eye_x, eye_y);
103
104             // Draw the new eyeCursor
105             eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
106             InvalidateRect(eyeCursor->getRect(),FALSE);
107         }
108         /* We count the number of gaze points at this button. We cannot rely on a fixation
109            message from detectFixation because we want the button to be clicked by looking
110            anywhere inside the button. gazeCount keeps track of the number of consecutive
111            samples within the button boundary. */
112         gazeCount++;
113
114         // After 30 samples press the button. Allow for 5 samples before releasing the button
115         // for a better visualization of a button click
116         if(gazeCount == 30){
117             isDown = true;
118             redraw();
119         }
120
121         // After 35 gaze samples consider the button clicked and notify the main app of the
122         // new state
123         if(gazeCount == 35){
124             isDown = false;
125
126             // Do appropriate action for button click
127             clicked();
128             redraw();
129         }
130
131         // Reset gazeCount so the button can be clicked again
132         if(gazeCount == 75){
133             gazeCount = 0;
134         }
135     }else{
136         // Set gazeCount back to 0 for this button
137         gazeCount=0;
138         if(isDown){
139             isDown = false;
140             redraw();
141         }
142         if(!once && !clutch){

```

```

143     // Erase the old eyecursor position
144     eyeCursor->erase(&buttontempDC);
145     InvalidateRect(eyeCursor->getRect(),FALSE);
146 }
147 }
148
149 return 0;
150 }
151
152 /*****
153 This function allows each type of button class to respond differently to a "click".
154 It must be defined in all derived classes.
155
156 1. Play the CLICK sound.
157 2. Post a message to the parent notifying it of the click.
158 *****/
159 void EyeDrawButton::clicked()
160 {
161     // after enough eye-samples, button gets clicked
162     // play clicking sound
163     if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
164     //post message to parent window notifying it of a click
165     CWnd *parent = GetParent();
166     parent->PostMessage(UWM_ED_STATE_CHANGE,value,type);
167 }
168
169 /*****
170 The redraw function redraws the button based on "up" or "down" states. Derived classes
171 can redefine this for a different appearance.
172
173 1. Determine if button is "up" or "down" and draw sunken or raised.
174 2. Draw the bitmap if this button has one.
175 *****/
176 void EyeDrawButton::redraw()
177 {
178     pMemDC = new CDC;
179     pMemDC->CreateCompatibleDC(&buttontempDC);
180
181     //Get the rectangle area of this button
182     this->GetClientRect(&button_rect);
183
184     //redraw grey square either sunken or raised
185     if ( (isDown) ) {
186         buttontempDC.FillSolidRect(button_rect, RGB_GREY);
187         buttontempDC.DrawEdge(button_rect,EDGE_SUNKEN,BF_RECT);
188     } else {
189         buttontempDC.FillSolidRect(button_rect, RGB_GREY); //grey
190         buttontempDC.DrawEdge(button_rect,EDGE_RAISED,BF_RECT);
191     }
192
193     // If there is a bitmap to draw on the button then do it
194     if (hasBitmap) {
195         CBitmap *oldBitmap = pMemDC->SelectObject(&bitmap);
196
197         buttontempDC.StretchBlt(button_rect.Width()/10,button_rect.Height()/10,button_rect.Width()*
198         4/5,button_rect.Height()*4/5,
199         pMemDC, 0, 0,100,
200         85,
201         SRCCOPY);
202         pMemDC->SelectObject(oldBitmap);
203     }
204 }
205
206 void EyeDrawButton::OnPaint()
207 {
208     // device context for painting
209     CPaintDC paintdc(this);
210
211

```

```

212 this->GetClientRect(&button_rect);
213
214 // The very first time OnPaint is called, itialize the device contexts
215 // and initialize the eye cursor, then draw everything needed.
216 if (once){
217     // Initialize mydc
218     mydc = GetDC();
219
220     // Initialize the dc for temporary background
221     buttontempDC.CreateCompatibleDC(mydc);
222     image.CreateCompatibleBitmap(mydc, button_rect.Width(), button_rect.Height());
223     buttontempDC.SelectObject(&image);
224     buttontempDC.FillSolidRect(&button_rect, RGB_GREY); // background
225     image.DeleteObject();
226
227     // Initialize the dc for permanent background
228     buttonmemoryDC.CreateCompatibleDC(mydc);
229     memoryRectBg.CreateCompatibleBitmap(mydc, button_rect.Width(), button_rect.Height());
230     buttonmemoryDC.SelectObject(&memoryRectBg);
231     buttonmemoryDC.FillSolidRect(&button_rect, RGB_GREY); // background
232
233     redraw();
234
235     // Initialize the eye cursor
236     eyeCursor = new EyeCursor(mydc,button_rect, RGB_GREY);
237     eyeCursor->setColor(RGB_BLUE);
238
239     eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
240     eyeCursor->erase(&buttontempDC);
241
242     InvalidateRect(eyeCursor->getRect(),FALSE);
243
244     // Set the timer to turn off the mouse after 7 seconds
245     this->SetTimer(4,7000,0);
246
247     once = false;
248 }
249
250 // Blt the temporary device context (because it contains everything) to screen
251 paintdc.BitBlt( button_rect.left,
252                 button_rect.top,
253                 button_rect.Width(),
254                 button_rect.Height(),
255                 &buttontempDC,
256                 0,
257                 0,
258                 SRCCOPY);
259 }
260
261 /*****
262 OnStateChange handles messages from EyeDrawDlg about changes in button state. If a
263 type of button wants to respond to this, they should define this function.
264
265 lParam represent the new state type
266 wParam represent the new value or change
267
268 1. Generic button does nothing.
269 *****/
270 LRESULT EyeDrawButton::OnStateChange(WPARAM wParam, LPARAM lParam){
271     //if the program's new state is the same as the button's type,
272     //then this tool has been selected.
273
274     return 0;
275 }
276
277 /*****
278 OnBitmapChange handles an message from EyeDrawDlg requesting a change is the button's
279 bitmap image.
280
281 lParam is the filename of the new bitmap.
282 wParam is not being used here.

```

```

283
284     1. Convert the filename into a LPTSTR.
285     2. Delete the old bitmap.
286     3. Load the new one.
287     4. Redraw the button.
288 *****/
289 LRESULT EyeDrawButton::OnBitmapChange(WPARAM wParam, LPARAM lParam){
290     // get the filename of the bitmap
291     filename = (CString *)lParam;
292
293     // These next two lines convert a filename into a LPTSTR so that it can be passed to
294     // LoadBMPImage
295     LPTSTR lpsz = new TCHAR[filename->GetLength()+1];
296     _tcsncpy(lpsz, *filename);
297
298     if(bitmap.m_hObject != NULL){
299         bitmap.DeleteObject();
300     }
301     // Load the bitmap from the file
302     hasBitmap = LoadBMPImage(lpsz, bitmap, NULL);
303
304     // redraw the button
305     redraw();
306     delete lpsz;
307
308     return 0;
309 }
310
311 /*****
312 This function gets called when mouse is moved. Usually the mouse is not visible,
313 so this function makes it visible for a short time because it was moved.
314
315 1. Change the mouse cursor to the standard arrow.
316 2. Set a timer so that the cursor will be turned off after a time of no movement.
317 *****/
318 void EyeDrawButton::OnMouseMove(UINT nFlags, CPoint point)
319 {
320     ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
321     this->SetTimer(4,7000,0);
322
323     CButton::OnMouseMove(nFlags, point);
324 }
325
326 /*****
327 This function gets called when timer started by a mouse move runs out. This means
328 that the mouse has been stationary for long enough to make it disappear again.
329
330 1. Change the mouse cursor to nothing (make it disappear).
331 *****/
332 void EyeDrawButton::OnTimer(UINT nIDEvent)
333 {
334     ::SetCursor(AfxGetApp()->LoadStandardCursor(0));
335
336     CButton::OnTimer(nIDEvent);
337 }
338
339 /*****
340 This function gets called when key from the keyboard is pressed. It is used to
341 simulate some of the appropriate actions of the F keys.
342
343 1. If it is an F1 or an F12, notify the parent.
344 *****/
345 void EyeDrawButton::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
346 {
347     if( nChar == VK_F1 || nChar == VK_F12 ){
348         CWnd *parent = GetParent();
349         parent->PostMessage(WM_KEYUP,nChar,nRepCnt);
350     }
351
352     CButton::OnKeyUp(nChar, nRepCnt, nFlags);
353 }

```

```

354 LRESULT EyeDrawButton::OnKeyUp( WPARAM wParam, LPARAM lParam )
355 {
356     if( wParam == VK_F1 || wParam == VK_F12 ){
357         CWnd *parent = GetParent();
358         parent->PostMessage(WM_KEYUP,wParam,lParam);
359     }
360     return 0;
361 }
362
363 ////////////////////////////////////////////////////
364 // EyeDraw Shape Button
365 ////////////////////////////////////////////////////
366
367 IMPLEMENT_DYNAMIC(ED_ShapeButton, EyeDrawButton)
368
369 // Standard Constructor
370 // - defaults are the button is not selected and the shape is not filled
371 ED_ShapeButton::ED_ShapeButton()
372 {
373     isToolSelected = false;
374     currentFill = false;
375 }
376
377 // Constructor taking
378 ED_ShapeButton::ED_ShapeButton(int value1)
379 {
380     type = SHAPE_STATE;           /* indicates which class it belongs to (SHAPE_STATE here)*/
381     value = value1;               /* indicates what it does if clicked */
382
383     // initially not selected
384     isToolSelected = false;
385     // initially not filled
386     currentFill = false;
387 }
388
389 // Destructor
390 ED_ShapeButton::~ED_ShapeButton(void){}
391
392
393 BEGIN_MESSAGE_MAP(ED_ShapeButton, EyeDrawButton)
394     // Message notifying button of change in program state
395     ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
396     // ON_WM_KEYUP()
397 END_MESSAGE_MAP()
398
399 /*****
400     This function allows shape buttons to respond differently to a "click". It is called
401     from within the base class's OnEyeMovement.
402
403     1. Play the CLICK sound.
404     2. Post a message to the parent notifying it of the click.
405     3. If this button is selected, toggle whether the shape is filled.
406     4. Notify the parent of the current fill status.
407 *****/
408 void ED_ShapeButton::clicked()
409 {
410     // after enough eye-samples, button gets clicked
411     // play clicking sound
412     if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
413     // post message to parent that button was clicked
414     CWnd *parent = GetParent();
415     // type = SHAPE_STATE, and value = what shape this button represents
416     parent->PostMessage(UWM_ED_STATE_CHANGE,value,type);
417
418     if(isToolSelected) {
419         // change the current fill after first click
420         currentFill = !currentFill;
421     }
422     // and tell the parent the fill has changed
423     parent->PostMessage(UWM_ED_STATE_CHANGE, currentFill, FILL);
424 }

```

```

425
426 /*****
427 The redraw function redraws the button as needed.
428
429 1. Call the base class's redraw to get the standard appearance.
430 2. Draw the shape corresponding to this button's tool and whether it is filled.
431 3. Resave the new button area into the location retrieved by the EyeCursor after it
432 moves. Otherwise, the EyeCursor will paste back the wrong image.
433 *****/
434 void ED_ShapeButton::redraw()
435 {
436 // call the base class's redraw
437 EyeDrawButton::redraw();
438
439 //draw the appropriate shape with the appropriate color (grey if not selected) and fill
440 CPen pen(PS_SOLID,5, isToolSelected ? currentColor : RGB_DARKGREY);
441 CPen* oldPen = buttontempDC.SelectObject(&pen);
442 COLORREF fill = (currentFill ? (isToolSelected ? currentColor : RGB_DARKGREY) :
... RGB_GREY);
443 CBrush brush(fill);
444 CBrush* oldBrush = (CBrush*)buttontempDC.SelectObject(&brush);
445 switch (value){
446 case FREE:
447     buttontempDC.MoveTo(10,10);
448     buttontempDC.LineTo(20,20);
449     buttontempDC.LineTo(15,25);
450     buttontempDC.LineTo(button_rect.Width()/2,button_rect.Height()/2);
451     buttontempDC.LineTo(button_rect.Width()/2 + 5, button_rect.Height()/2 + 10);
452     buttontempDC.LineTo(button_rect.Width()/2 + 10, button_rect.Height()/2 + 15);
453     buttontempDC.LineTo(15, 25);
454     buttontempDC.LineTo(button_rect.Width() - 15, button_rect.Height()/2 - 20);
455     buttontempDC.LineTo(button_rect.Width() - 10, button_rect.Height()/2 - 10);
456     buttontempDC.LineTo(button_rect.Width()/2 - 5, 10);
457     buttontempDC.LineTo(button_rect.Width() - 10, button_rect.Height() - 10);
458     break;
459 case LINE:
460     if(currentFill){
461         buttontempDC.MoveTo(40,20);
462         buttontempDC.LineTo(20,button_rect.Height()-20);
463         buttontempDC.LineTo(button_rect.Width()-20,button_rect.Height()-20);
464         buttontempDC.LineTo(button_rect.Width()/2,button_rect.Height()/2);
465         buttontempDC.LineTo(button_rect.Width() - 20, button_rect.Height()/2 - 20);
466     }else{
467         buttontempDC.MoveTo(20,20);
468         buttontempDC.LineTo(button_rect.Width() - 20, button_rect.Height() - 20);
469     }
470     break;
471 case CIRCLE:
472     if ( isToolSelected ){
473         buttontempDC.Ellipse(10,20,button_rect.Width()-10,button_rect.Height()-20);
474     }else{
475         buttontempDC.Ellipse(10,20,button_rect.Width()-10,button_rect.Height()-20);
476     }
477     break;
478 case RECTANGLE:
479     if ( isToolSelected ){
480         buttontempDC.Rectangle(20,20,button_rect.Width()-20,button_rect.Height()-20);
481     }else{
482         buttontempDC.Rectangle(20,20,button_rect.Width()-20,button_rect.Height()-20);
483     }
484     break;
485 }
486 buttontempDC.SelectObject(oldPen);
487 buttontempDC.SelectObject(oldBrush);
488
489 // And copy the tempDC to the memoryDC
490 buttonmemoryDC.BitBlt( button_rect.left,
491     button_rect.top,
492     button_rect.Width(),
493     button_rect.Height(),
494     &buttontempDC,

```

```

495         0,
496         0,
497         SRCCOPY);
498
499     if(!once){
500         // Draw the new eyeCursor
501         eyeCursor->draw(&buttonTempDC, &buttonMemoryDC);
502         InvalidateRect(eyeCursor->getRect(),FALSE);
503         // Erase the old eyeCursor position
504         eyeCursor->erase(&buttonTempDC);
505         InvalidateRect(eyeCursor->getRect(),FALSE);
506
507         InvalidateRect(button_rect,FALSE);
508     }
509
510     InvalidateRect(button_rect,FALSE);
511 }
512
513
514 /*****
515     OnStateChange handles messages from EyeDrawDlg about changes in button state.
516
517     lParam represent the new state type
518     wParam represent the new value or change
519
520     Shape buttons only care about the following conditions:
521     1. If the new state type is SHAPE_STATE and the value is this button's value, this
522        button is now selected. If its not this button's value, it is deselected.
523     2. If the new state type is a change in color, change this button's color to reflect
524        that.
525     3. If the new state type is a stamp, this button can't possibly be selected.
526 *****/
527 LRESULT ED_ShapeButton::OnStateChange(WPARAM wParam, LPARAM lParam)
528 {
529     //if the program's new state is the same as the button's type,
530     //then this tool has been selected.
531     if((int)lParam == SHAPE_STATE){
532         if ( value == (int)wParam){
533             if(!isToolSelected){
534                 isToolSelected = true;
535                 redraw();
536             }
537         }else{
538             if(isToolSelected){
539                 isToolSelected = false;
540                 redraw();
541             }
542         }
543     }
544     // if the message is of a new color, reset the button's shape color
545     if( (int)lParam == COLOR ){
546         currentColor = (COLORREF)wParam;
547         redraw();
548     }
549     if( (int)lParam == STAMP_STATE ){
550         if(isToolSelected){
551             isToolSelected = false;
552             redraw();
553         }
554     }
555
556     return 0;
557 }
558
559 ///////////////////////////////////////////////////
560 // EyeDraw Color Button
561 ///////////////////////////////////////////////////
562
563 IMPLEMENT_DYNAMIC(ED_ColorButton, EyeDrawButton)
564
565 // Standard Constructor sets the button's color to white and not selected.

```

```

566 ED_ColorButton::ED_ColorButton()
567 {
568     color = RGB(255,255,255);
569     isColorSelected = false;
570 }
571
572 // Constructor taking a value and a color.
573 ED_ColorButton::ED_ColorButton(int value1, COLORREF color1)
574 {
575     type = COLOR;           /* indicates the class it belongs to, determined by color*/
576     value = value1;         /* indicates what it does if clicked */
577     color = color1;
578     // The black color button is the initial program default
579     if (color == RGB_BLACK) {
580         isColorSelected = true;
581     } else {
582         isColorSelected = false;
583     }
584 }
585
586 // Destructor
587 ED_ColorButton::~ED_ColorButton(void){}
588
589 BEGIN_MESSAGE_MAP(ED_ColorButton, EyeDrawButton)
590     // Message notifying button of change in program state
591     ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
592 END_MESSAGE_MAP()
593
594 /*****
595     This function allows color buttons to respond differently to a "click". It is called
596     from within the base class's OnEyeMovement.
597
598     1. Play the CLICK sound.
599     2. Post a message to the parent notifying it of the click.
600 *****/
601 void ED_ColorButton::clicked()
602 {
603     // play clicking sound
604     if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
605     // post message to parent that button was clicked
606     CWnd *parent = GetParent();
607     // type = COLOR_STATE, and value = what color this button represents
608     parent->PostMessage(UWM_ED_STATE_CHANGE,color,type);
609 }
610
611 /*****
612     The redraw function redraws the button as needed.
613
614     1. Call the base class's redraw to get the standard appearance.
615     2. Draw a rectangle that is small if not selected and larger if the button is selected.
616     3. Resave the new button area into the location retrieved by the EyeCursor after it
617         moves. Otherwise, the EyeCursor will paste back the wrong image.
618 *****/
619 void ED_ColorButton::redraw()
620 {
621     // Call the base class's DrawItem
622     EyeDrawButton::redraw();
623
624     // Draw the rectangle with this button's color, either small or large
625     if (isColorSelected) {
626
627         ... buttontempDC.FillSolidRect(button_rect.Width()/8,button_rect.Height()/8,button_rect.Width()
628         ... *3/4,button_rect.Height()*3/4,color);
629     } else {
630
631         ... buttontempDC.FillSolidRect(button_rect.Width()/3,button_rect.Height()/3,button_rect.Width()
632         ... /3,button_rect.Height()/3,color);
633     }
634
635     // And copy the tempDC to the memoryDC
636     buttonmemoryDC.BitBlt( button_rect.left,

```

```

633         button_rect.top,
634         button_rect.Width(),
635         button_rect.Height(),
636         &buttontempDC,
637         0,
638         0,
639         SRCCOPY);
640
641     if(!once){
642         // Draw the new eyeCursor
643         eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
644         InvalidateRect(eyeCursor->getRect(),FALSE);
645         // Erase the old eyecursor position
646         eyeCursor->erase(&buttontempDC);
647         InvalidateRect(eyeCursor->getRect(),FALSE);
648
649         InvalidateRect(button_rect,FALSE);
650     }
651 }
652
653 }
654
655 /*****
656     OnStateChange handles messages from EyeDrawDlg about changes in button state.
657
658     lParam represent the new state type
659     wParam represent the new value or change
660
661     Color buttons only care about the following conditions:
662     1. If the new state type is COLOR and the color is this button's color, this
663        button is now selected. If its not this button's color, it is deselected.
664 *****/
665 LRESULT ED_ColorButton::OnStateChange(WPARAM wParam, LPARAM lParam){
666
667     // if the program's new state is the same as the button's type,
668     // then this tool has been selected.
669     if( (int)lParam == COLOR ){
670         if( color == (COLORREF)wParam ) {
671             if(!isColorSelected){
672                 isColorSelected = true;
673                 redraw();
674             }
675         }else{
676             if(isColorSelected){
677                 isColorSelected = false;
678                 redraw();
679             }
680         }
681     }
682
683     return 0;
684 }
685
686 //////////////////////////////////////
687 // EyeDraw Utility Button
688 //////////////////////////////////////
689
690 IMPLEMENT_DYNAMIC(ED_UtilityButton, EyeDrawButton)
691
692 // Standard Constructor with generic defaults
693 ED_UtilityButton::ED_UtilityButton()
694 {
695     gazeCount = 0;           /* number of times eye gaze samples on the button */
696     isDown = false;         /* starts in the up state */
697     name = "";              /* default text is nothing */
698 }
699
700 // Constructor taking a state and two texts for when button is selected and deselected.
701 ED_UtilityButton::ED_UtilityButton(int state, CString selectedName1, CString
... deselectedName1, COLORREF fontcolor)
702 {

```

```

703 gazeCount = 0; /* number of times eye gaze samples on the button */
704
705 isDown = false; /* starts in the up state */
706 isToolSelected = false; /* starts out deselected */
707 hasBitmap = false; /* originally no bitmaps are displayed on button */
708 type = state; /* the type of button */
709 value = 0;
710 selectedName = selectedName1; /* the text to display when the button is selected */
711 deselectedName = deselectedName1; /* the text to display when the button is deselected*/
712 name = deselectedName1; /* show the deselected name first */
713 color = fontcolor; /* font color for this button */
714 clutch = false; /* the "Dot" button is not yet selected */
715
716 }
717
718 // Destructor
719 ED_UtilityButton::~ED_UtilityButton(){}
720
721
722 BEGIN_MESSAGE_MAP(ED_UtilityButton, EyeDrawButton)
723 // Message notifying button of change in program state
724 ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
725
726 // Message requesting the button change the text written on it.
727 ON_MESSAGE(UWM_ED_CAPTION_CHANGE, OnCaptionChange)
728 END_MESSAGE_MAP()
729
730 /*****
731 This function allows utility buttons to respond differently to a "click". It is called
732 from within the base class's OnEyeMovement.
733
734 1. Play the CLICK sound.
735 2. Post a message to the parent notifying it of the click.
736 *****/
737 void ED_UtilityButton::clicked()
738 {
739 // after enough eye-samples, button gets clicked
740 // play clicking sound
741 if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
742 // post message to parent that button was clicked
743 CWnd *parent = GetParent();
744 // type = the state change that this button represents, and value is not being used in
745 // this case
746 parent->PostMessage(UWM_ED_STATE_CHANGE,value,type);
747 }
748
749 /*****
750 The redraw function redraws the button as needed.
751
752 1. Call the base class's redraw to get the standard appearance.
753 2. Use built in fonts and sizes to draw the name on the button. The name will already
754 have the appropriate text.
755 3. Resave the new button area into the location retrieved by the EyeCursor after it
756 moves. Otherwise, the EyeCursor will paste back the wrong image.
757 *****/
758 void ED_UtilityButton::redraw()
759 {
760 // call the base class's DrawItem
761 EyeDrawButton::redraw();
762
763 this->GetClientRect(&button_rect);
764
765 // if this is a grid button, and it is selected, draw the grid on it
766 CPen pen(PS_SOLID,1,RGB_BLACK);
767 CPen* oldPen = buttontempDC.SelectObject(&pen);
768 if(type == GRID_STATE){
769 for (int i = button_rect.Width()/8; i <= 7*button_rect.Width()/8;
... i+=button_rect.Width()/8){
770 for (int j = button_rect.Height()/8; j < 7*button_rect.Height()/8;
... j+=button_rect.Height()/8){
771 buttontempDC.Ellipse(i,j,i+2,j+2);

```

```

772     }
773 }
774 }
775 buttontempDC.SelectObject(oldPen);
776
777 // create a font and draw the button's text (name)
778 CFont font;
779
... font.CreateFont(button_rect.Height()/3,button_rect.Width()/10,0,0,FW_BOLD,0,0,0,0,0,0,0,0,"
... Arial");
780 CFont* def_font = buttontempDC.SelectObject(&font);
781 buttontempDC.SetBkColor(RGB_GREY);
782 buttontempDC.SetTextColor(color);
783 buttontempDC.DrawText(name, button_rect, DT_CENTER|DT_VCENTER|DT_SINGLELINE);
784 buttontempDC.SelectObject(def_font);
785 font.DeleteObject();
786
787 // And copy the tempDC to the memoryDC
788 buttonmemoryDC.BitBlt( button_rect.left,
789                        button_rect.top,
790                        button_rect.Width(),
791                        button_rect.Height(),
792                        &buttontempDC,
793                        0,
794                        0,
795                        SRCCOPY);
796
797 if(!once){
798     // Draw the new eyeCursor
799     eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
800     InvalidateRect(eyeCursor->getRect(),FALSE);
801     // Erase the old eyecursor position
802     eyeCursor->erase(&buttontempDC);
803     InvalidateRect(eyeCursor->getRect(),FALSE);
804
805     InvalidateRect(button_rect,FALSE);
806 }
807
808 InvalidateRect(button_rect,FALSE);
809
810 if(clutch){
811     // If clutch is true, this means that the only button receiving eye movement data
812     // is the "Dot On"/"Dot Off" button, so
813     // Park the eyeCursor
814     eyeCursor->move(button_rect.left + button_rect.Width()/2 - 5, button_rect.top +
... button_rect.Height()/3 - 5);
815
816     // Draw the new eyeCursor
817     eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
818     InvalidateRect(eyeCursor->getRect(),FALSE);
819 }
820 }
821
822 /*****
823 OnCaptionChange handles messages requesting a change in the text written on the button.
824
825 wParam and lParam are not being used here.
826
827 1. Toggle the selectedName and deselectedName.
828 2. Redraw the button.
829 *****/
830 LRESULT ED_UtilityButton::OnCaptionChange(WPARAM wParam, LPARAM lParam){
831     // changes the caption of the button when clicked
832     if(isToolSelected){
833         name.SetString(deselectedName);
834         isToolSelected = false;
835         clutch = false;
836     }else{ //isToolSelected == false
837         name.SetString(selectedName);
838         isToolSelected = true;
839         if(type == DOT_STATE){

```

```

840         clutch = true;
841     }
842 }
843 redraw();
844 return 0;
845 }
846
847 //////////////////////////////////////
848 // EyeDraw Stamp Button
849 //////////////////////////////////////
850
851 IMPLEMENT_DYNAMIC(ED_StampButton, EyeDrawButton)
852
853 // Standard constructor with default values
854 ED_StampButton::ED_StampButton()
855 {
856     gazeCount = 0;                /* number of times eye gaze samples on the button */
857     isDown = false;               /* button starts in "up" position */
858     hasBitmap = false;           /* stamp buttons have resource ids, not bitmaps */
859 }
860
861 // Constructor taking an id for the resource id of the button's bitmap
862 ED_StampButton::ED_StampButton(int id){
863     gazeCount = 0;                /* number of times eye gaze samples on the button */
864     isDown = false;               /* starts in the up state */
865     type = STAMP;                 /* the type of button */
866     hasBitmap = false;           /* stamp buttons have resource ids, not bitmaps */
867     resourceID = id;             /* resource id of this button's bitmap */
868 }
869
870 // Destructor
871 ED_StampButton::~ED_StampButton(){}
872
873 BEGIN_MESSAGE_MAP(ED_StampButton, EyeDrawButton)
874     // Message notifying button of change in program state
875     ON_MESSAGE(UWM_ED_STATE_CHANGE, OnStateChange)
876 END_MESSAGE_MAP()
877
878 /*****
879     This function allows stamp buttons to respond differently to a "click". It is called
880     from within the base class's OnEyeMovement.
881
882     1. Play the CLICK sound.
883     2. Post a message to the parent notifying it of the click.
884 *****/
885 void ED_StampButton::clicked()
886 {
887     // after enough eye-samples, button gets clicked
888     // play clicking sound
889     if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
890     // post message to parent that button was clicked
891     CWnd *parent = GetParent();
892     // type = the state change that this button represents, and value is not being used in
893     // this case
894     parent->PostMessage(UWM_ED_STATE_CHANGE, resourceID, type);
895 }
896
897 /*****
898     The redraw function redraws the button as needed.
899
900     1. Draw the button based on "up" or "down" as raised or sunken.
901     2. Draw the stamp bitmap with a transparent background on the button.
902     3. Resave the new button area into the location retrieved by the EyeCursor after it
903         moves.
904     Otherwise, the EyeCursor will paste back the wrong image.
905 *****/
906 void ED_StampButton::redraw()
907 {
908     EyeDrawButton::redraw();
909
910     CBitmap bmp;

```

```

911 bmp.LoadBitmap(resourceID);
912
913 HBITMAP hBitmap= (HBITMAP) bmp.GetSafeHandle();
914
915 // use a temporary DC just so that we can stretch blt it to the real DC
916 CDC *tmpDC = this->GetDC();
917
918 tmpDC->FillSolidRect(0,0,70,70,RGB_GREY);
919
920 DrawTransparentBitmap( *tmpDC,    // The destination DC.
921                        hBitmap,    // The bitmap to be drawn.
922                        0,          // X coordinate.
923                        0,          // Y coordinate.
924                        0x00FFFFFF); // The color for transparent
925                                // pixels (white, in this example).
926
927 // Then stretch it to fit the button
928 buttontempDC.StretchBlt(button_rect.Width()/8,
929                          button_rect.Height()/8,
930                          3*button_rect.Width()/4,
931                          3*button_rect.Height()/4,
932                          tmpDC,0,0,70,70,SRCCOPY);
933
934 if(isToolSelected){
935     //draw a green box around it
936     CPen pen(PS_SOLID, 4, RGB_GREEN);
937
938     // Select the pen into the device context.
939     CPen* pOldPen = (CPen*)buttontempDC.SelectObject(&pen);
940
941     buttontempDC.MoveTo(button_rect.left+10,button_rect.top+10);
942     buttontempDC.LineTo(button_rect.right-10,button_rect.top+10);
943     buttontempDC.LineTo(button_rect.right-10,button_rect.bottom-10);
944     buttontempDC.LineTo(button_rect.left+10,button_rect.bottom-10);
945     buttontempDC.LineTo(button_rect.left+10,button_rect.top+10);
946
947     //Restore original device context object
948     buttontempDC.SelectObject(pOldPen);
949
950 }
951
952 // And copy the tempDC to the memoryDC
953 buttonmemoryDC.BitBlt( button_rect.left,
954                        button_rect.top,
955                        button_rect.Width(),
956                        button_rect.Height(),
957                        &buttontempDC,
958                        0,
959                        0,
960                        SRCCOPY);
961
962 if(!once){
963     // Draw the new eyeCursor
964     eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
965     InvalidateRect(eyeCursor->getRect(),FALSE);
966     // Erase the old eyecursor position
967     eyeCursor->erase(&buttontempDC);
968     InvalidateRect(eyeCursor->getRect(),FALSE);
969
970     InvalidateRect(button_rect,FALSE);
971 }
972
973 InvalidateRect(button_rect,FALSE);
974 }
975
976 /*****
977 OnStateChange handles messages from EyeDrawDlg about changes in button state.
978
979 lParam represent the new state type
980 wParam represent the new value or change
981

```

```

982     Stamp buttons only care about the following conditions:
983     1. If the new state type is STAMP_STATE and the value is this button's value, this
984         button is now selected. If its not this button's value, it is deselected.
985     *****/
986 LRESULT ED_StampButton::OnStateChange(WPARAM wParam, LPARAM lParam)
987 {
988     //if the program's new state is the same as the button's type,
989     //then this tool has been selected.
990     if((int)lParam == STAMP){
991         if ( resourceID == (int)wParam){
992             if(!isToolSelected){
993                 isToolSelected = true;
994                 redraw();
995             }
996         }else{
997             if(isToolSelected){
998                 isToolSelected = false;
999                 redraw();
1000             }
1001         }
1002     }
1003
1004     return 0;
1005 }
1006
1007 //
1008 // http://support.microsoft.com/default.aspx?scid=http://support.microsoft.com:80/support/kb/
1009 // articles/Q79/2/12.asp&NoWebContent=1
1010 void ED_StampButton::DrawTransparentBitmap(HDC hdc, HBITMAP hBitmap, short xStart,
1011     short yStart, COLORREF cTransparentColor)
1012 {
1013     BITMAP        bm;
1014     COLORREF      cColor;
1015     HBITMAP       bmAndBack, bmAndObject, bmAndMem, bmSave;
1016     HBITMAP       bmBackOld, bmObjectOld, bmMemOld, bmSaveOld;
1017     HDC           hdcMem, hdcBack, hdcObject, hdcTemp, hdcSave;
1018     POINT         ptSize;
1019
1020     hdcTemp = CreateCompatibleDC(hdc);
1021     SelectObject(hdcTemp, hBitmap);    // Select the bitmap
1022
1023     GetObject(hBitmap, sizeof(BITMAP), (LPSTR)&bm);
1024     ptSize.x = bm.bmWidth;             // Get width of bitmap
1025     ptSize.y = bm.bmHeight;            // Get height of bitmap
1026     DPTOLP(hdcTemp, &ptSize, 1);      // Convert from device
1027                                     // to logical points
1028
1029     // Create some DCs to hold temporary data.
1030     hdcBack  = CreateCompatibleDC(hdc);
1031     hdcObject = CreateCompatibleDC(hdc);
1032     hdcMem   = CreateCompatibleDC(hdc);
1033     hdcSave  = CreateCompatibleDC(hdc);
1034
1035     // Create a bitmap for each DC. DCs are required for a number of
1036     // GDI functions.
1037
1038     // Monochrome DC
1039     bmAndBack  = CreateBitmap(ptSize.x, ptSize.y, 1, 1, NULL);
1040
1041     // Monochrome DC
1042     bmAndObject = CreateBitmap(ptSize.x, ptSize.y, 1, 1, NULL);
1043
1044     bmAndMem    = CreateCompatibleBitmap(hdc, ptSize.x, ptSize.y);
1045     bmSave      = CreateCompatibleBitmap(hdc, ptSize.x, ptSize.y);
1046
1047     // Each DC must select a bitmap object to store pixel data.
1048     bmBackOld   = (HBITMAP)SelectObject(hdcBack, bmAndBack);
1049     bmObjectOld = (HBITMAP)SelectObject(hdcObject, bmAndObject);
1050     bmMemOld    = (HBITMAP)SelectObject(hdcMem, bmAndMem);
1051     bmSaveOld   = (HBITMAP)SelectObject(hdcSave, bmSave);

```

```

1051 // Set proper mapping mode.
1052 SetMapMode(hdcTemp, GetMapMode(hdc));
1053
1054 // Save the bitmap sent here, because it will be overwritten.
1055 BitBlt(hdcSave, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0, SRCCOPY);
1056
1057 // Set the background color of the source DC to the color.
1058 // contained in the parts of the bitmap that should be transparent
1059 cColor = SetBkColor(hdcTemp, cTransparentColor);
1060
1061 // Create the object mask for the bitmap by performing a BitBlt
1062 // from the source bitmap to a monochrome bitmap.
1063 BitBlt(hdcObject, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0,
1064        SRCCOPY);
1065
1066 // Set the background color of the source DC back to the original
1067 // color.
1068 SetBkColor(hdcTemp, cColor);
1069
1070 // Create the inverse of the object mask.
1071 BitBlt(hdcBack, 0, 0, ptSize.x, ptSize.y, hdcObject, 0, 0,
1072        NOTSRCCOPY);
1073
1074 // Copy the background of the main DC to the destination.
1075 BitBlt(hdcMem, 0, 0, ptSize.x, ptSize.y, hdc, xStart, yStart,
1076        SRCCOPY);
1077
1078 // Mask out the places where the bitmap will be placed.
1079 BitBlt(hdcMem, 0, 0, ptSize.x, ptSize.y, hdcObject, 0, 0, SRCAND);
1080
1081 // Mask out the transparent colored pixels on the bitmap.
1082 BitBlt(hdcTemp, 0, 0, ptSize.x, ptSize.y, hdcBack, 0, 0, SRCAND);
1083
1084 // XOR the bitmap with the background on the destination DC.
1085 BitBlt(hdcMem, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0, SRCPAINT);
1086
1087 // Copy the destination to the screen.
1088 BitBlt(hdc, xStart, yStart, ptSize.x, ptSize.y, hdcMem, 0, 0,
1089        SRCCOPY);
1090
1091 // Place the original bitmap back into the bitmap sent here.
1092 BitBlt(hdcTemp, 0, 0, ptSize.x, ptSize.y, hdcSave, 0, 0, SRCCOPY);
1093
1094 // Delete the memory bitmaps.
1095 DeleteObject(SelectObject(hdcBack, bmBackOld));
1096 DeleteObject(SelectObject(hdcObject, bmObjectOld));
1097 DeleteObject(SelectObject(hdcMem, bmMemOld));
1098 DeleteObject(SelectObject(hdcSave, bmSaveOld));
1099
1100 // Delete the memory DCs.
1101 DeleteDC(hdcMem);
1102 DeleteDC(hdcBack);
1103 DeleteDC(hdcObject);
1104 DeleteDC(hdcSave);
1105 DeleteDC(hdcTemp);
1106 }
1107
1108 ///////////////////////////////////////////////////
1109 // EyeDraw Key Button
1110 ///////////////////////////////////////////////////
1111
1112 IMPLEMENT_DYNAMIC(ED_KeyButton, EyeDrawButton)
1113
1114 // Standard Constructor with generic defaults
1115 ED_KeyButton::ED_KeyButton()
1116 {
1117     gazeCount = 0; // number of times eye gaze samples on the button */
1118     isDown = false; // starts in the up state */
1119     value = ""; // default text is nothing */
1120 }

```

```

1122
1123 // Constructor taking a state and two texts for when button is selected and deselected.
1124 ED_KeyButton::ED_KeyButton(CString value1)
1125 {
1126     gazeCount = 0;                /* number of times eye gaze samples on the button */
1127
1128     isDown = false;                /* starts in the up state */
1129     hasBitmap = false;            /* originally no bitmaps are displayed on button */
1130     type = KEY;                   /* the type of button */
1131     value = value1;
1132     clutch = false;              /* the "Dot" button is not yet selected */
1133 }
1134
1135 // Destructor
1136 ED_KeyButton::~~ED_KeyButton(){}
1137
1138 BEGIN_MESSAGE_MAP(ED_KeyButton, EyeDrawButton)
1139 END_MESSAGE_MAP()
1140
1141 /*****
1142 This function allows key buttons to respond differently to a "click". It is called
1143 from within the base class's OnEyeMovement.
1144
1145 1. Play the CLICK sound.
1146 2. Post a message to the parent notifying it of the click.
1147 *****/
1148 void ED_KeyButton::clicked()
1149 {
1150     // after enough eye-samples, button gets clicked
1151     // play clicking sound
1152     if(Sound) PlaySound("EyeDraw/Sounds/CLICK.WAV", NULL, SND_FILENAME | SND_ASYNC);
1153     // post message to parent that button was clicked
1154     CWnd *parent = GetParent();
1155     // type = the state change that this button represents, and value is not being used in
1156     // this case
1157     parent->PostMessage(UWM_ED_STATE_CHANGE, (WPARAM)&value, type);
1158 }
1159
1160 /*****
1161 The redraw function redraws the button as needed.
1162
1163 1. Call the base class's redraw to get the standard appearance.
1164 2. Use built in fonts and sizes to draw the name on the button.
1165 3. Resave the new button area into the location retrieved by the EyeCursor after it
1166 moves.
1167 Otherwise, the EyeCursor will paste back the wrong image.
1168 *****/
1169 void ED_KeyButton::redraw()
1170 {
1171     // call the base class's DrawItem
1172     EyeDrawButton::redraw();
1173
1174     this->GetClientRect(&button_rect);
1175
1176     if(isDown) buttontempDC.FillSolidRect(button_rect, RGB_GREEN);
1177     // create a font and draw the button's text (name)
1178     CFont font;
1179     font.CreateFont(button_rect.Height()/3, 10, 0, 0, FW_BOLD, 0, 0, 0, 0, 0, 0, 0, 0, "Arial");
1180     CFont* def_font = buttontempDC.SelectObject(&font);
1181     buttontempDC.SetBkColor( isDown? RGB_GREEN : RGB_GREY);
1182     buttontempDC.DrawText(value, button_rect, DT_CENTER|DT_VCENTER|DT_SINGLELINE);
1183     buttontempDC.SelectObject(def_font);
1184     font.DeleteObject();
1185
1186     // And copy the tempDC to the memoryDC
1187     buttonmemoryDC.BitBlt( button_rect.left,
1188         button_rect.top,
1189         button_rect.Width(),
1190         button_rect.Height(),

```

```

1193         &buttontempDC,
1194         0,
1195         0,
1196         SRCCOPY);
1197
1198     if(!once){
1199         // Draw the new eyeCursor
1200         eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
1201         InvalidateRect(eyeCursor->getRect(),FALSE);
1202         // Erase the old eyecursor position
1203         eyeCursor->erase(&buttontempDC);
1204         InvalidateRect(eyeCursor->getRect(),FALSE);
1205
1206         InvalidateRect(button_rect,FALSE);
1207     }
1208
1209     InvalidateRect(button_rect,FALSE);
1210
1211     if(clutch){
1212         // If clutch is true, this means that the only button receiving eye movement data
1213         // is the "Dot On"/"Dot Off" button, so
1214         // Park the eyeCursor
1215         eyeCursor->move(button_rect.left + button_rect.Width()/2 - 5, button_rect.top +
1216         ... button_rect.Height()/3 - 5);
1217
1218         // Draw the new eyeCursor
1219         eyeCursor->draw(&buttontempDC, &buttonmemoryDC);
1220         InvalidateRect(eyeCursor->getRect(),FALSE);
1221     }
1222 }

```