Name: Rodrigo Ignacio Rojas Garcia
Course: ECE 4310
Lab #4

Region Interaction

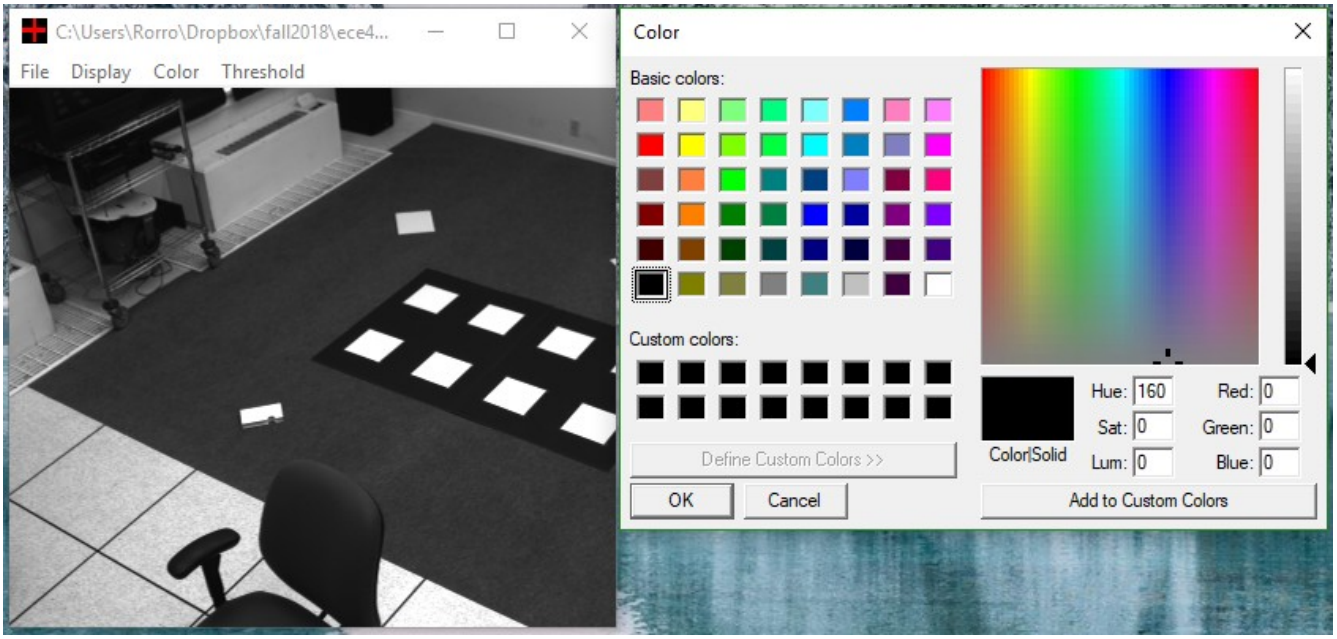
In this project the student was to implement interactive region growing. The student was provided with a built in Visual Studio project called “plus” which demonstrated several GUI and event handling techniques and region growing C code which demonstrated the region grow based on several predicates. The student was to use these two given pieces in order to complete the following requirements:

1. Program to visualize the region growing by coloring pixels as they join the growing region.
    1. Program should have a GUI option which allowed the user to select the color for pixels to join the region.
    2. Program should have an option that clears the result of a previous region grow, displaying the original image.
2. Program should have a menu option to grow the region in “Play” mode or in “Step” mode.
    1. In “Play” mode, a pixel should join the region each 1 ms.
    2. In “Step” mode, a pixel should join the region each time the user presses the key ‘j’.
    3. The program should allow the user to change between modes while a region is growing.
3. Program should use a dialog box to allow user to select values for two predicates for joining a region.
    1. First predicated is the absolute difference of the pixel intensity to the average intensity of pixels already in the region (To join, a pixel must be within this range).
    2. Second predicate is the distance of the pixel to the centroid of pixels already in the region (To join, a pixel must be within this range).
    3. Both predicates should be applied at the same time while growing the region.

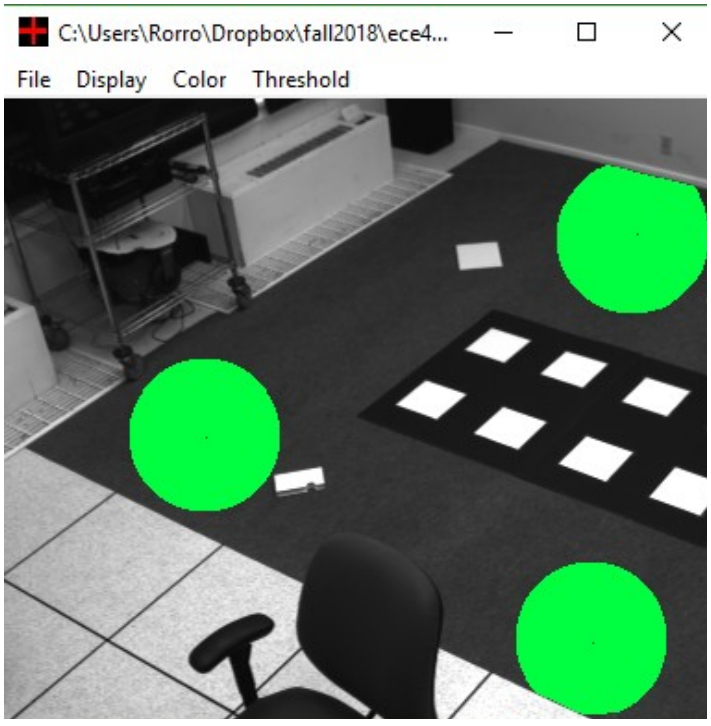
All C code can be seen at the end of the report.

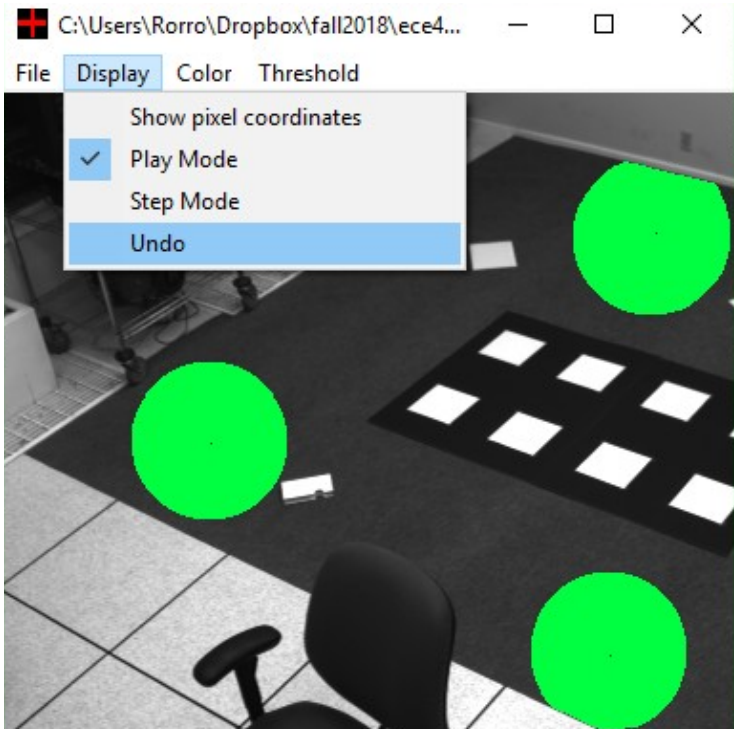
REQUIREMENT I

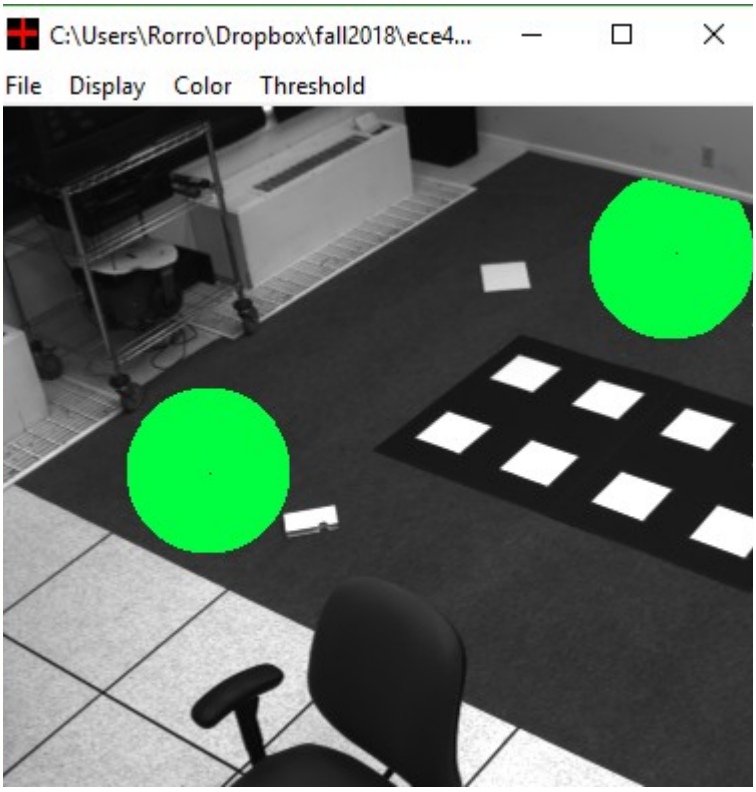
GUI allowing user to choose a color from Color Palette:



Program showing option to clear the result of previous region grow:

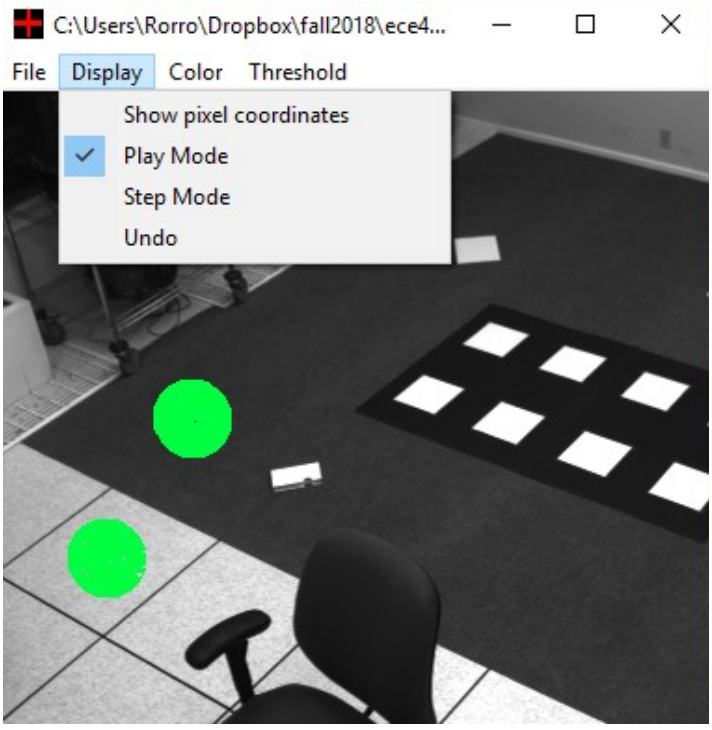




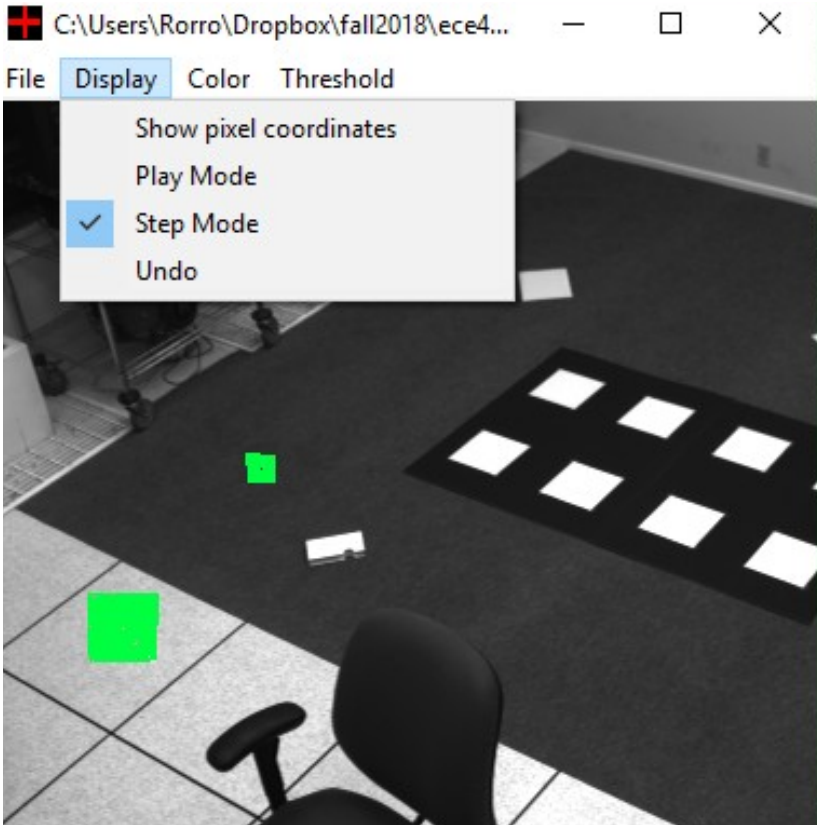


REQUIREMENT II

Program showing "Play" mode

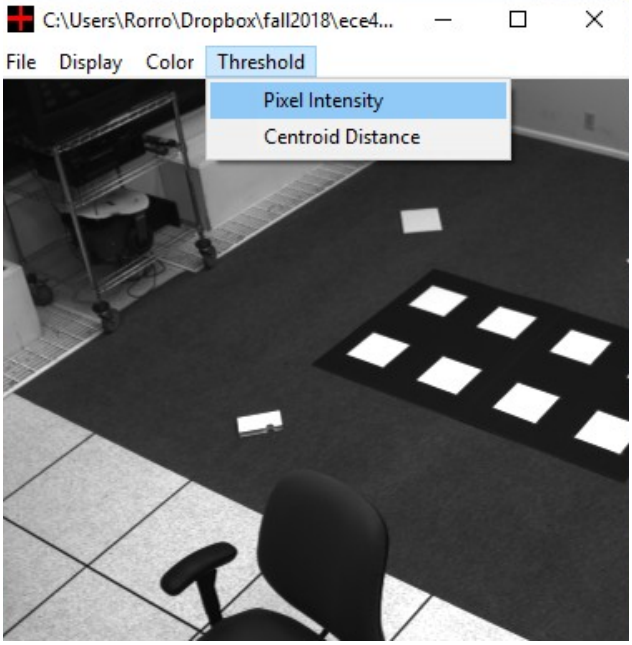


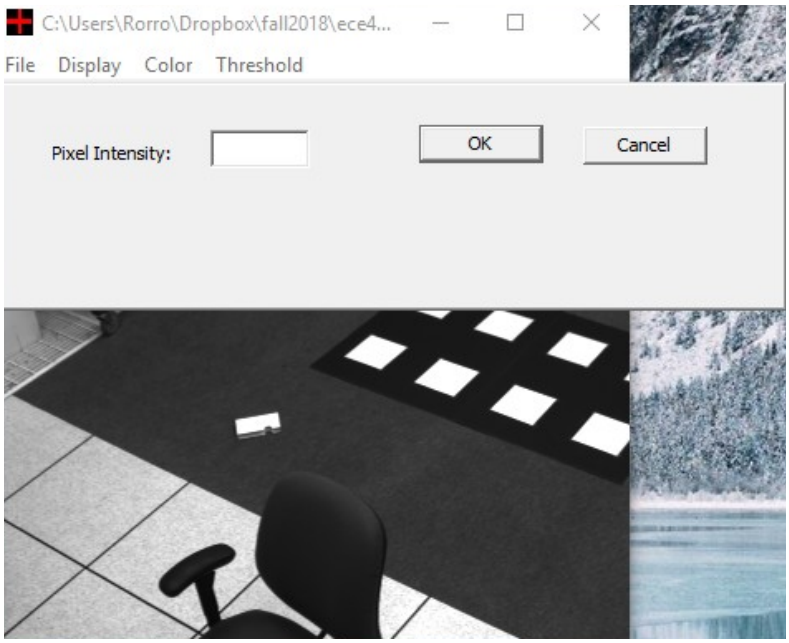
Program showing “Step” mode



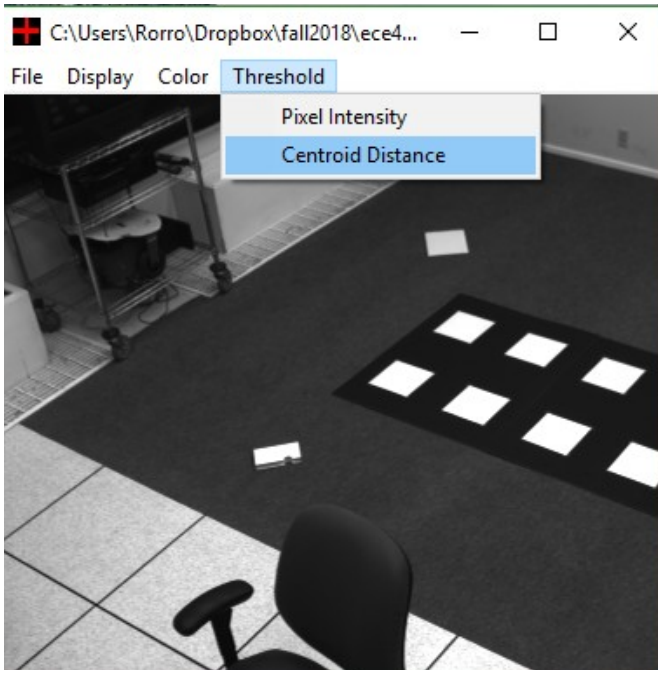
REQUIREMENT III

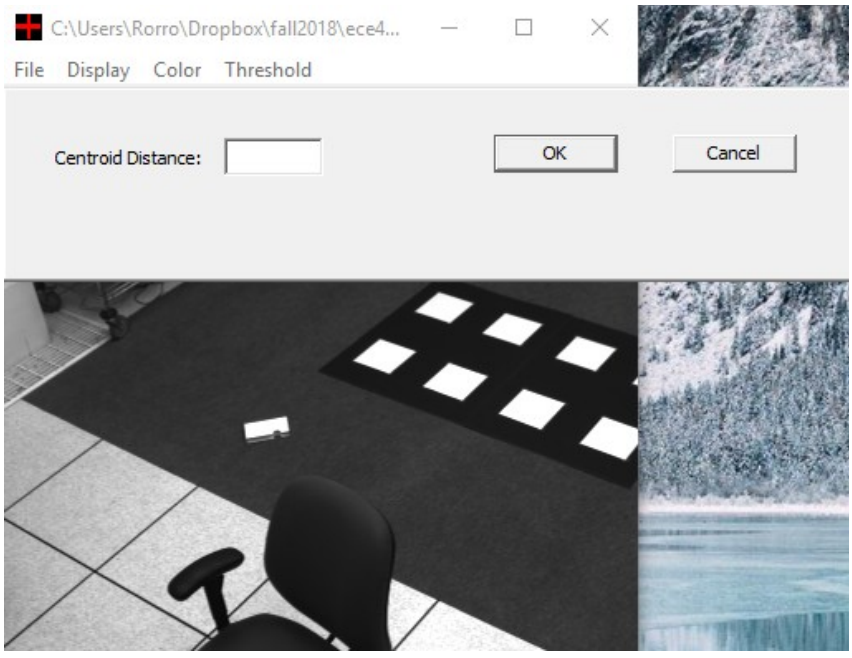
Program showing Pixel Intensity Dialog Box :



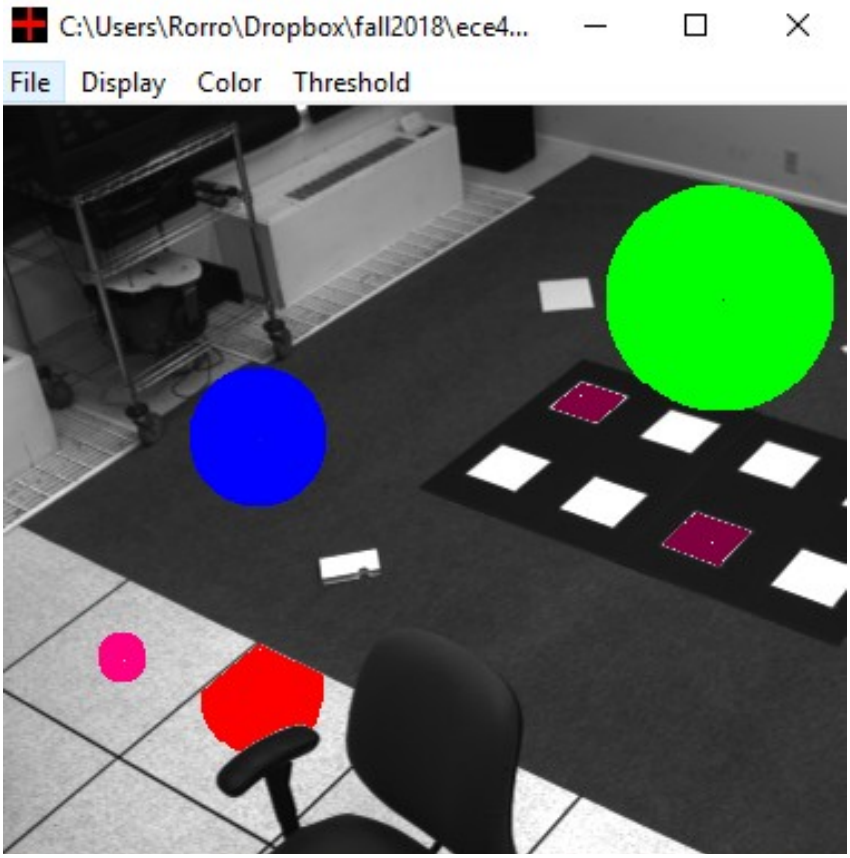


Program showing Centroid Distance Dialog Box:





RESULTS


C:\Users\Rorro\Dropbox\fall2018\ece4...
File
Display
Color
Threshold

```c
/*
    Name: Rodrigo Ignacio Rojas Garcia
    Lab#: 4
*/

// Library Declaration Section
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <sys/timeb.h>
#include <windows.h>
#include <wingdi.h>
#include <winuser.h>
#include <process.h>
#include "resource.h"
#include "globals.h"

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine,
int nCmdShow)
{
    MSG         msg;
    HWND        hWnd;
    WNDCLASS    wc;

    wc.style=CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc=(WNDPROC)WndProc;
    wc.cbClsExtra=0;
    wc.cbWndExtra=0;
    wc.hInstance=hInstance;
    wc.hIcon=LoadIcon(hInstance,"ID_PLUS_ICON");
    wc.hCursor=LoadCursor(NULL,IDC_ARROW);
    wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName="ID_MAIN_MENU";
    wc.lpszClassName="PLUS";

    if (!RegisterClass(&wc))
    {
        return(FALSE);
    }

    hWnd=CreateWindow("PLUS","plus program", WS_OVERLAPPEDWINDOW | WS_HSCROLL |
    WS_VSCROLL, CW_USEDEFAULT,0,400,400,NULL,NULL,hInstance,NULL);
    if (!hWnd)
    {
        return(FALSE);
    }

    ShowScrollBar(hWnd,SB_BOTH,FALSE);
    ShowWindow(hWnd,nCmdShow);
    UpdateWindow(hWnd);
    MainWnd=hWnd;

    /* SETS GLOBAL VARIABLES ONCE PROGRAM STARTS */
    ShowPixelCoords=0;
    play_mode = 0;
    step_mode = 0;
    total_threads = 0;

    strcpy(filename,"");
    OriginalImage=NULL;
    ROWS=COLS=0;

    InvalidateRect(hWnd,NULL,TRUE);
    UpdateWindow(hWnd);

    while (GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
```

```
    }

    return(msg.wParam);
}

/* CALLBACK FOR PIXEL INTENSITY*/
LRESULT CALLBACK WndProc2(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
    case WM_COMMAND:
        switch (LOWORD(wParam))
        {
            case IDOK:
                GetDlgItemText(hWnd, IDC_EDIT1, threshold, 256);
                thresh = atoi(threshold);
                EndDialog(hWnd, wParam);
                break;
            case IDCANCEL:
                EndDialog(hWnd, wParam);
        }
    }
    return(0L);
}

/* CALLBACK FOR CENTROID DISNTACE */
LRESULT CALLBACK WndProc3(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
    case WM_COMMAND:
        switch (LOWORD(wParam))
        {
        case IDOK:
            GetDlgItemText(hWnd, IDC_EDIT1, radius, 256);
            rad = atoi(radius);
            EndDialog(hWnd, wParam);
            break;
        case IDCANCEL:
            EndDialog(hWnd, wParam);
        }
    }
    return(0L);
}

LRESULT CALLBACK WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    // Variable Declaration Section
    HMENU               hMenu;
    OPENFILENAME        ofn;
    FILE                *fpt;
    HDC                 hDC;
    char                header[320],text[320];
    int                 BYTES,xPos,yPos;

    switch (uMsg)
    {
        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
                /* EVENT WHEN USER SELECT SHOW PIXEL */
                case ID_SHOWPIXELCOORDS:
                    ShowPixelCoords = (ShowPixelCoords + 1) % 2;
                    //PaintImage();
                    break;

                /* EVENT WHEN USER SELECTS PLAY MODE*/
                case ID_PLAY_MODE:
                    play_mode = (play_mode + 1) % 2;
```

```
137                     step_mode = 0;
138                     end_thread = 0;
139                     break;
140
141                 /* EVENT WHEN USER SELECTS STEP MODE*/
142                 case ID_STEP_MODE:
143                     step_mode = (step_mode + 1) % 2;
144                     play_mode = 0;
145                     end_thread = 0;
146                     break;
147
148                 /* CREATES DIALOG BOX FOR COLORS*/
149                 case ID_SELECT_COLOR:
150                     ZeroMemory(&color, sizeof(CHOOSECOLOR));
151                     color.lStructSize = sizeof(CHOOSECOLOR);
152                     color.hwndOwner = hWnd;
153                     color.lpCustColors = (LPDWORD)acrCustClr;
154                     color.rgbResult = rgbCurrent;
155                     color.Flags = CC_FULLOPEN | CC_RGBINIT;
156                     if (ChooseColor(&color) == TRUE)
157                     {
158                         hbrush = CreateSolidBrush(color.rgbResult);
159                         rgbCurrent = color.rgbResult;
160                     }
161                     break;
162
163                 /* EVENT WHEN USER SELECTS UNDO */
164                 case ID_UNDO:
165                     end_thread = 1;
166                     int i, j;
167                     HDC hDC;
168                     /* UNDO'S LAST THREAD DRAWING BY SETTING PIXELS CHANGED TO ORIGINAL
                        IMAGE RGB VALUES*/
169                     for (i = 0; i < ROWS; i++)
170                     {
171                         for (j = 0; j < COLS; j++)
172                         {
173                             if (indices[i * COLS + j] == total_threads)
174                             {
175                                 hDC = GetDC(MainWnd);
176                                 SetPixel(hDC, j, i, RGB(OriginalImage[i * COLS + j],
177                                     OriginalImage[i * COLS + j], OriginalImage[i * COLS
                                        + j]));
178                                 ReleaseDC(MainWnd, hDC);
179                                 indices[i * COLS + j] = 0;
180                             }
181                         }
182                     }
183                     end_thread = 0;
184                     if (total_threads > 0)
185                     {
186                         total_threads -= 1;
187                     }
188                     break;
189
190                 /* EVENT WHEN USER SELECTS TO CHANGE PIXEL INTENSITY*/
191                 case ID_PIXEL_INTENSITY:
192                     DialogBox(NULL, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, WndProc2);
193                     break;
194
195                 /* EVENT WHEN USER SELECTS TO CHANGE THE CENTROID DISTANCE */
196                 case ID_CENTROID_DISTANCE:
197                     DialogBox(NULL, MAKEINTRESOURCE(IDD_DIALOG2), hWnd, WndProc3);
198                     break;
199
200                 /* EVENT WHEN USER SELECTS TO LOAD A PICTURE TO THE GUI*/
201                 case ID_FILE_LOAD:
202                     if (OriginalImage != NULL)
203                     {
```

```
204                     free(OriginalImage);
205                     OriginalImage = NULL;
206                 }
207                 memset(&(ofn), 0, sizeof(ofn));
208                 ofn.lStructSize = sizeof(ofn);
209                 ofn.lpstrFile = filename;
210                 filename[0] = 0;
211                 ofn.nMaxFile = MAX_FILENAME_CHARS;
212                 ofn.Flags = OFN_EXPLORER | OFN_HIDEREADONLY;
213                 ofn.lpstrFilter = "PPM files\0*.ppm\0All files\0*.*\0\0";
214                 if (!(GetOpenFileName(&ofn)) || filename[0] == '\0')
215                 {
216                     break;      /* user cancelled load */
217                 }
218                 if ((fpt = fopen(filename, "rb")) == NULL)
219                 {
220                     MessageBox(NULL, "Unable to open file", filename, MB_OK |
                    MB_APPLMODAL);
221                     break;
222                 }
223                 fscanf(fpt, "%s %d %d %d", header, &COLS, &ROWS, &BYTES);
224                 if (strcmp(header, "P5") != 0 || BYTES != 255)
225                 {
226                     MessageBox(NULL, "Not a PPM (P5 greyscale) image", filename,
                    MB_OK | MB_APPLMODAL);
227                     fclose(fpt);
228                     break;
229                 }
230                 OriginalImage = (unsigned char *)calloc(ROWS*COLS, 1);
231                 header[0] = fgetc(fpt); /* whitespace character after header */
232                 fread(OriginalImage, 1, ROWS*COLS, fpt);
233                 fclose(fpt);
234                 SetWindowText(hWnd, filename);
235                 PaintImage();
236
237                 /* CREATES A GLOBAL VARIABLE FOR INDICES*/
238                 indices = (int *)calloc(ROWS * COLS, sizeof(int));
239
240                 break;
241
242             case ID_FILE_QUIT:
243                 DestroyWindow(hWnd);
244                 break;
245         }
246         break;
247
248     case WM_SIZE:          /* could be used to detect when window size changes */
249         PaintImage();
250         return(DefWindowProc(hWnd,uMsg,wParam,lParam));
251         break;
252
253     case WM_PAINT:
254         PaintImage();
255         return(DefWindowProc(hWnd,uMsg,wParam,lParam));
256         break;
257
258     case WM_LBUTTONDOWN:case WM_RBUTTONDOWN:
259
260         /* MOUSE CLICK FOR PLAY MODE OR STEP MODE */
261         if ((play_mode == 1) || (step_mode == 1))
262         {
263             /* GETS THE X AND Y POSITION OF CLICKED POSITION IN IMAGE*/
264             mouse_x_pos = LOWORD(lParam);
265             mouse_y_pos = HIWORD(lParam);
266
267             /* CREATES A THREAD AND BEGINS REGION GROW ON IMAGE */
268             if ((mouse_x_pos >= 0) && (mouse_x_pos < COLS) && (mouse_y_pos >= 0) &&
                (mouse_y_pos < ROWS));
269             {
```

```
                    fill_thread_running = 1;
                    _beginthread(region_grow, 0, MainWnd);
                    total_threads += 1;
                }

            }
            return(DefWindowProc(hWnd,uMsg,wParam,lParam));
            break;

        /* EVENT THAT SHOWS CURRENT CURSOR PIXEL AND DRAWS ON IMAGE */
        case WM_MOUSEMOVE:
            if (ShowPixelCoords == 1)
            {
                  xPos=LOWORD(lParam);
                  yPos=HIWORD(lParam);
                  if (xPos >= 0  &&  xPos < COLS  &&  yPos >= 0  &&  yPos < ROWS)
                  {
                        sprintf(text,"%d, %d => %d", xPos, yPos,
                        OriginalImage[yPos*COLS+xPos]);
                        hDC=GetDC(MainWnd);
                        TextOut(hDC,0,0,text,strlen(text));     /* draw text on the
                        window */
                        SetPixel(hDC,xPos,yPos,RGB(255,0,0));   /* color the cursor
                        position red */
                        ReleaseDC(MainWnd,hDC);
                  }
            }
        return(DefWindowProc(hWnd,uMsg,wParam,lParam));
        break;

        case WM_KEYDOWN:
            if (wParam == 's' || wParam == 'S')
            {
                PostMessage(MainWnd, WM_COMMAND, ID_SHOWPIXELCOORDS, 0);      /* send
                message to self */
            }
            /* VENT THAT TRIGGERS WHEN USER PRESSES J ON STEP MODE, THE REGION KEEPS
            GROWING */
            if (wParam == 'j' || wParam == 'J')
            {
                is_j_pressed = 1;
            }
            if ((TCHAR)wParam == '1')
            {
                TimerRow=TimerCol=0;
                SetTimer(MainWnd,TIMER_SECOND,10,NULL); /* start up 10 ms timer */
            }
            if ((TCHAR)wParam == '2')
            {
                KillTimer(MainWnd,TIMER_SECOND);            /* halt timer, stopping
                generation of WM_TIME events */
                PaintImage();                               /* redraw original image,
                erasing animation */
            }
            if ((TCHAR)wParam == '3')
            {
                ThreadRunning = 1;
                _beginthread(AnimationThread,0,MainWnd);    /* start up a child thread
                to do other work while this thread continues GUI */
            }
            if ((TCHAR)wParam == '4')
            {
                ThreadRunning = 0;
            }
            return(DefWindowProc(hWnd,uMsg,wParam,lParam));
            break;

        case WM_TIMER:    /* this event gets triggered every time the timer goes off */
            hDC=GetDC(MainWnd);
```

```
331             SetPixel(hDC,TimerCol,TimerRow,RGB(0,0,255));   /* color the animation
                pixel blue */
332             ReleaseDC(MainWnd,hDC);
333             TimerRow++;
334             TimerCol+=2;
335             break;
336
337         case WM_HSCROLL:      /* this event could be used to change what part of the
                image to draw */
338             PaintImage();     /* direct PaintImage calls eliminate flicker; the
                alternative is InvalidateRect(hWnd,NULL,TRUE); UpdateWindow(hWnd); */
339             return(DefWindowProc(hWnd,uMsg,wParam,lParam));
340             break;
341
342         case WM_VSCROLL:      /* this event could be used to change what part of the
                image to draw */
343             PaintImage();
344             return(DefWindowProc(hWnd,uMsg,wParam,lParam));
345             break;
346
347         case WM_DESTROY:
348             PostQuitMessage(0);
349             break;
350         default:
351             return(DefWindowProc(hWnd,uMsg,wParam,lParam));
352             break;
353     }
354
355     hMenu=GetMenu(MainWnd);
356
357     /* CHECKS AND UNCHECKS IF SELECTED OPTION ON GUI FOR SHOWING PIXEL COORDINATES
358         PLAY MODE, AND STEP MODE */
359     if (ShowPixelCoords == 1)
360     {
361         CheckMenuItem(hMenu, ID_SHOWPIXELCOORDS, MF_CHECKED);   /* you can also call
                EnableMenuItem() to grey(disable) an option */
362     }
363     else
364     {
365         CheckMenuItem(hMenu, ID_SHOWPIXELCOORDS, MF_UNCHECKED);
366     }
367     if (play_mode == 1)
368     {
369         CheckMenuItem(hMenu, ID_PLAY_MODE, MF_CHECKED);
370     }
371     else
372     {
373         CheckMenuItem(hMenu, ID_PLAY_MODE,  MF_UNCHECKED);
374     }
375     if (step_mode == 1)
376     {
377         CheckMenuItem(hMenu, ID_STEP_MODE, MF_CHECKED);
378     }
379     else
380     {
381         CheckMenuItem(hMenu, ID_STEP_MODE, MF_UNCHECKED);
382     }
383
384     DrawMenuBar(hWnd);
385
386     return(0L);
387 }
388
389
390 /* FUNCTION GOES BACK TO ORIGINAL IMAGE */
391 void PaintImage()
392
393 {
394     PAINTSTRUCT         Painter;
```

```
395       HDC                 hDC;
396       BITMAPINFOHEADER    bm_info_header;
397       BITMAPINFO          *bm_info;
398       int                 i,r,c,DISPLAY_ROWS,DISPLAY_COLS;
399       unsigned char       *DisplayImage;
400
401       if (OriginalImage == NULL)
402       {
403           return;     /* no image to draw */
404       }
405
406       /* Windows pads to 4-byte boundaries.  We have to round the size up to 4 in each
          dimension, filling with black. */
407       DISPLAY_ROWS = ROWS;
408       DISPLAY_COLS = COLS;
409
410       if (DISPLAY_ROWS % 4 != 0)
411       {
412           DISPLAY_ROWS = (DISPLAY_ROWS / 4 + 1) * 4;
413       }
414       if (DISPLAY_COLS % 4 != 0)
415       {
416           DISPLAY_COLS = (DISPLAY_COLS / 4 + 1) * 4;
417       }
418
419       DisplayImage = (unsigned char *)calloc(DISPLAY_ROWS*DISPLAY_COLS,1);
420
421       for (r = 0; r < ROWS; r++)
422       {
423           for (c = 0; c < COLS; c++)
424           {
425               DisplayImage[r*DISPLAY_COLS + c] = OriginalImage[r*COLS + c];
426           }
427       }
428
429       BeginPaint(MainWnd,&Painter);
430       hDC=GetDC(MainWnd);
431       bm_info_header.biSize=sizeof(BITMAPINFOHEADER);
432       bm_info_header.biWidth=DISPLAY_COLS;
433       bm_info_header.biHeight=-DISPLAY_ROWS;
434       bm_info_header.biPlanes=1;
435       bm_info_header.biBitCount=8;
436       bm_info_header.biCompression=BI_RGB;
437       bm_info_header.biSizeImage=0;
438       bm_info_header.biXPelsPerMeter=0;
439       bm_info_header.biYPelsPerMeter=0;
440       bm_info_header.biClrUsed=256;
441       bm_info_header.biClrImportant=256;
442       bm_info=(BITMAPINFO *)calloc(1,sizeof(BITMAPINFO) + 256*sizeof(RGBQUAD));
443       bm_info->bmiHeader=bm_info_header;
444
445       for (i=0; i<256; i++)
446       {
447
              bm_info->bmiColors[i].rgbBlue=bm_info->bmiColors[i].rgbGreen=bm_info->bmiColors
              [i].rgbRed=i;
448           bm_info->bmiColors[i].rgbReserved=0;
449       }
450
451       SetDIBitsToDevice(hDC,0,0,DISPLAY_COLS,DISPLAY_ROWS,0,0,0, /* first scan line
          */DISPLAY_ROWS, /* number of scan lines */DisplayImage,bm_info,DIB_RGB_COLORS);
452       ReleaseDC(MainWnd,hDC);
453       EndPaint(MainWnd,&Painter);
454
455       free(DisplayImage);
456       free(bm_info);
457   }
458
459
```

```
460  void AnimationThread(HWND AnimationWindowHandle)
461  {
462      HDC     hDC;
463      char    text[300];
464
465      ThreadRow = ThreadCol = 0;
466      while (ThreadRunning == 1)
467      {
468            hDC=GetDC(MainWnd);
469            SetPixel(hDC,ThreadCol,ThreadRow,RGB(0,255,0));   /* color the animation
               pixel green */
470            sprintf(text,"%d, %d",ThreadRow,ThreadCol);
471            TextOut(hDC,300,0,text,strlen(text));     /* draw text on the window */
472            ReleaseDC(MainWnd,hDC);
473            ThreadRow+=3;
474            ThreadCol++;
475            Sleep(100);       /* pause 100 ms */
476      }
477  }
478
479  /* REGION GROW FUNCTION */
480  void region_grow(HWND play_mode_window_handle)
481  {
482      // Variable Declaration Section
483      HDC hDC;
484      int row, col;
485      int x_pos = mouse_x_pos;
486      int y_pos = mouse_y_pos;
487      int has_been_painted = 0;
488      int queue[MAX_QUEUE];
489      int queue_head, queue_tail;
490      int average;
491      int total;
492      int count;
493      int index;
494      int i;
495      int x_1, y_1, x_2, y_2;
496      int distance;
497      int thread_num = total_threads;
498      unsigned char *labels;
499
500      labels = (unsigned char *)calloc(ROWS * COLS, sizeof(unsigned char));
501
502      average = total = (int)OriginalImage[y_pos*COLS + x_pos];
503
504      index = (y_pos * COLS) + x_pos;
505      labels[index] = 1;
506      queue[0] = index;
507
508      queue_head = 1;
509      queue_tail = 0;
510      count = 1;
511      is_j_pressed = 0;
512
513      x_1 = x_pos; // Original X value for Centroid COLS
514      y_1 = y_pos; // Original Y value for Centroid ROWS
515      x_2 = x_pos;
516      y_2 = y_pos;
517
518
519      while (queue_head != queue_tail && fill_thread_running == 1)
520      {
521          hDC = GetDC(MainWnd);
522          // Recalculate the average every 50 pixels
523          if ((count % 50) == 0)
524          {
525              average = total / count;
526          }
527          /* STOPS ALL THREADS IF STEP MODE AND PLAY MODE ARE NOT SELECTED */
```

```
528         while (step_mode == 0 && play_mode == 0)
529         {
530             if (end_thread == 1 && (total_threads == thread_num))
531             {
532                 _endthread();
533             }
534         }
535         for (row = -1; row <= 1; row++)
536         {
537             for (col = -1; col <= 1; col++)
538             {
539                 if ((row == 0) && (col == 0))
540                 {
541                     continue;
542                 }
543                 if ((queue[queue_tail] / COLS + row) < 0 ||
544                     (queue[queue_tail] / COLS + row) >= ROWS ||
545                     (queue[queue_tail] % COLS + col) < 0 ||
546                     (queue[queue_tail] % COLS + col) >= COLS)
547                 {
548                     continue;
549                 }
550                 if (labels[(queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
                    COLS + col] != has_been_painted)
551                 {
552                     continue;
553                 }
554
555                 /* KILLS LAST CREATED THREAD*/
556                 if (end_thread == 1 && (total_threads == thread_num))
557                 {
558                     _endthread();
559                 }
560
561                 /* TEST CRITERIA TO SEE IF PIXEL CAN BE JOINED */
562                 index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
                    COLS + col;
563                 if (abs((int)(OriginalImage[index] - average)) > thresh)
564                 {
565                     continue;
566                 }
567
568                 /* CENTROID CRITERIA */
569                 x_2 = x_1 / count; // COL
570                 y_2 = y_1 / count; // ROW
571                 distance = sqrt(SQR((queue[queue_tail] / COLS + row) - y_2) +
                    SQR((queue[queue_tail] % COLS + col) - x_2));
572                 if (distance > rad)
573                 {
574                     continue;
575                 }
576                 x_1 += queue[queue_tail] % COLS + col;
577                 y_1 += queue[queue_tail] / COLS + row;
578
579                 /* PAINTS CURRENT IMAGE BASED ON SELECTED PIXELS AND SELECTED COLOR */
580                 SetPixel(hDC, queue[queue_tail] % COLS + col, queue[queue_tail] / COLS
                    + row,
581                     RGB(GetRValue(rgbCurrent), GetGValue(rgbCurrent),
                        GetBValue(rgbCurrent)));
582
583                 /* IF PIXEL HAS BEEN PAINTED, THREAD NUMBER WHICH PAINTED IT IS SAVED
                    TO ARRAY FOR UNDO PURPOSES */
584                 if (indices[(queue[queue_tail] / COLS + row) * COLS +
                    (queue[queue_tail] % COLS + col)] == 0)
585                 {
586                     indices[(queue[queue_tail] / COLS + row) * COLS +
                        (queue[queue_tail] % COLS + col)] = thread_num;
587                 }
588
```

```
589                 /* LABELS KEEPS TRACK IF PIXEL HAS BEEN PAINTED OR NOT IN ORDER TO SKIP
                    IF IT HAS*/
590                 index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
                    COLS + col;
591                 labels[index] = 1;
592
593                 total += OriginalImage[index];
594                 count++;
595
596                 index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
                    COLS + col;
597                 queue[queue_head] = index;
598                 queue_head = (queue_head + 1) % MAX_QUEUE;
599
600                 if (queue_head == queue_tail)
601                 {
602                     exit(0);
603                 }
604
605                 /* DIFFERENCES BETWEEN PROCEDURE OF PLAY MODE AND STEP MODE */
606                 if (play_mode == 1)
607                 {
608                     Sleep(1);
609                 }
610                 /* STUCK ON WHILE LOOP WHILE USER DOES NOT PRESS J ON STEP MODE */
611                 else
612                 {
613                     while ((is_j_pressed == 0) && (step_mode == 1)) {}
614                     Sleep(1);
615                     is_j_pressed = 0;
616                 }
617
618             }
619         }
620         queue_tail = (queue_tail + 1) % MAX_QUEUE;
621         ReleaseDC(MainWnd, hDC);
622     }
623     _endthread();
624 }
625
626
```

```
1
2   #define SQR(x) ((x)*(x))    /* macro for square */
3   #ifndef M_PI            /* in case M_PI not found in math.h */
4   #define M_PI 3.1415927
5   #endif
6   #ifndef M_E
7   #define M_E 2.718282
8   #endif
9
10  #define MAX_FILENAME_CHARS  320
11
12  char    filename[MAX_FILENAME_CHARS];
13
14  HWND    MainWnd;
15
16          // Display flags
17  int     ShowPixelCoords;
18  int play_mode;
19  int step_mode;
20
21          // Image data
22  unsigned char   *OriginalImage;
23  int             ROWS, COLS;
24
25  #define TIMER_SECOND    1           /* ID of timer used for animation */
26  #define MAX_QUEUE 10000
27
28          // Drawing flags
29  int     TimerRow,TimerCol;
30  int     ThreadRow,ThreadCol;
31  int     ThreadRunning;
32
33  int mouse_x_pos, mouse_y_pos;
34  int is_j_pressed;
35  int fill_thread_running;
36  int red, green, blue;
37  HWND temp;
38  CHOOSECOLOR color;
39  DWORD rgbCurrent;
40  COLORREF acrCustClr[16];
41  HBRUSH hbrush;
42  int end_thread;
43  char threshold[256];
44  char radius[256];
45  int thresh;
46  int rad;
47  int total_threads;
48  int *indices;
49
50          // Function prototypes
51  LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
52  void PaintImage();
53  void AnimationThread(void *);       /* passes address of window */
54  void region_grow(void *); // Passes address of window
```