```
1  using System;
2  using System.Drawing;
3  using System.Drawing.Imaging;
4  using System.Drawing.Printing;
5  using System.Windows.Forms;
6  using System.Runtime.InteropServices;
7  using System.Text.RegularExpressions;
8  using System.Linq;
9  using System.Collections.Generic;
10 using System.IO;
11
12 //Author: James Rogers, St. Norbert College '18, Computer Science, Graphic
     Design Double Major
13 //Class: CSCI 460
14 //Completed over the Spring 2018 semester
15 //Assistance and guidance from Dr. Dave Pankratz and Dr. Bonita McVey
16
17 namespace Capstone4602._0._1
18 {
19
20     /// <summary>
21     /// Capstone Project Version 2.0.1 - Illustrating Text By Tag Clouds
22     /// This program maily utilizes two controls in Windows Forms - Labels and
         the GroupBox
23     /// The main goal of the program is to interpret textual data and place it
         into a tag cloud
24     /// in a way that is pleasing to the eye and downloadable.
25     /// The tag cloud is customizeable in a number of ways - Font, Shape, and
         Color.
26     /// How the process works - The main path for the program flow
27     /// 1. Data read in from two separate .txt sources that are tied together -
         can be found in debug folder
28     /// 2. User hits the RUN button
29     /// 3. Program creates empty labels within GroupBox according to quantity
         given by the user (starts at max of 300)(generateLabel())
30     /// 4. Program places data into each label according to quantity
31     /// 5. Program places labels into the groupbox according to random placement
          and zone increments (arrange label(generateLabel()))
32     /// 5.5 Program will recursively call placement function until labels are
         placed
33     /// 6. Program condenses labels towards center (condense())
34     /// 7. once all words are placed, user has opportunity to save tag cloud
35     /// 8. User then also has opportunity to change tag cloud shape, color,
         font, and data source
36     /// </summary>
37
38     public partial class Form1 : Form
39     {
40         //NOTE AFTER PRESENTATION - CHANGE SIZE OF FONT BASED ON WORD QUANTITY
41         //Lower quantity = larger font size, Larger quantity = smaller font size
42
43         ///////////////GLOBAL VARIABLES////////////////
```

```
44
45          //boolean to detect whether user has changed to a shaped background
46          bool ShapeBack = false;
47          //global image variables to initialize a bitmap only when user needs it
48          Bitmap map;
49          Image defaultBackgroundImage;
50          Image original;
51          //factor to multiply word size
52          int fontMult = 7;
53          //integer to identify which source the words are coming from
54          public int SourceChoice = 1;
55          //for checking on words printed
56          public int totalWordQuantity = 0;
57          public int MaxWords = 299;
58          //keeping track of collisions and changing the placement zone
59          public int totalMisses = 0;
60          public int setMisses = 0;
61          public int Increment = 10;  //seems to work best to split group box into
               10 increments
62          public int incHeight = 28;  //comes from cutting the halfway zone making
               it about ten increments until the boarder is reached
63          public int incWidth = 42;   //same as above
64          //to make the label exist in the first place
65          public int xCoord = 1;
66          public int yCoord = 1;
67          public Point hold;
68          //when the algorithm was more random based
69          public const int MAXFONT = 35; //not used anymore
70          public const int MINFONT = 2;  //not used anymore
71          //dont let words go past bounds of GroupBox
72          public int MINWIDTH = 5;
73          public int MINHEIGHT = 5;
74          public int MAXWIDTH = 850;
75          public int MAXHEIGHT = 580;
76          //needed for random placement
77          Random rnd = new Random();
78          //color array - should make more user customizable
79          public Color[] ColorArray = new Color[] { Color.Crimson, Color.Blue,
              Color.MediumPurple };
80          //font array
81          public string[] FontArray = new string[] {"Stencil", "Times New Roman",
              "Microsoft Sans Serif", "Wingdings", "Impact", "Harlow Solid Italic" ,
               "Cooper Black" , "Comic Sans MS"};
82          public string MasterFont = "Stencil";
83          //Very important!! keeps track of place in word and number arrays
84          public int ArrayPlace = 0;
85          //declared data arrays which are to be loaded in by readInstances()
86          string[] WordArray = new string[300]; //actual words
87          int[] QuantityArray = new int[300];   //quantities of words
88
89          //Win32 needed for pixel color catching, within getColorFromScreen
              (point)
```

```
 90          public class Win32
 91          {
 92              [DllImport("user32.dll")]
 93              static extern IntPtr GetDC(IntPtr hwnd);
 94
 95              [DllImport("user32.dll")]
 96              static extern Int32 ReleaseDC(IntPtr hwnd, IntPtr hdc);
 97
 98              [DllImport("gdi32.dll")]
 99              static extern uint GetPixel(IntPtr hdc, int nXPos, int nYPos);
100
101              //very important function for detecting color within shape for
                   shaped tag clouds
102              static public Color GetPixelColor(int x, int y)
103              {
104                  IntPtr hdc = GetDC(IntPtr.Zero);
105                  uint pixel = GetPixel(hdc, x, y);
106                  ReleaseDC(IntPtr.Zero, hdc);
107                  Color color = Color.FromArgb((int)(pixel & 0x000000FF),
108                                (int)(pixel & 0x0000FF00) >> 8,
109                                (int)(pixel & 0x00FF0000) >> 16);
110                  return color;
111              }
112          }
113
114
115          public Form1()
116          {
117              InitializeComponent();
118              //centers the window to the screen so that the screen capture does
                   not grab anything else
119              CenterToScreen();
120              //sets the background image to blank automatically
121              defaultBackgroundImage = canvasBox.BackgroundImage;
122              //reads the default data file immediately
123              ReadInstances(SourceChoice);
124
125          }
126          private Label GenerateLabel()
127          {   //Make a Label as needed by feeder function
128              Label x = new Label();
129              //adds the label to the groupBox
130              this.canvasBox.Controls.Add(x);
131              //@ location 1,1(top left corner)
132              hold.X = xCoord;
133              hold.Y = yCoord;
134              x.Location = hold;
135              //AutoSize must be enabled for least whitespace
136              x.AutoSize = true;
137              //Checks if a background shape is present
138              if (ShapeBack)
139              {
```

```
140                 x.BackColor = Color.White;
141             }
142             //creates toolTip which displays the count of each word when the
                  mouse hovers over a label
143             x.MouseHover += (s, e) =>
144             {
145                 ToolTip cms = new ToolTip();
146                 cms.Show(QuantityArray[x.TabIndex].ToString(), canvasBox,
                      x.Location, 2000);
147             };
148             //returns the label to the ArrangeLabel Function
149             return x;
150         }
151         private void ArrangeLabel(Label x)
152         {
153             //To arrange given Label with information across groupbox, in a
                  randomized way
154
155             //must figure a way out to not let Labels lay on top of other
                  existing Labels // 2/27/18
156             //figured way to check interaction with other Labels // 2/27/18 -
                  hours later
157
158             //Must soon implement a way that is not random i.e. spiral,
                  rectangular
159             //random placement takes too long
160             //not practical, fonts should start to get smaller faster with data
                  from twitter crawler or other source
161             // checks if button collides with another button, or the wall of the
                   group box itself 2/28/18
162
163             // Changes made to font size algorithm based off quantity!! Finally
                  4/4/18
164             // looks much better
165             // Stencil is best font for readability and the least amount of
                  wasted space
166
167             // need to choose between algorithm of purely quantity based versus
                  using the Log() function
168
169             // Chose to use the Log algorithm as it looks most appealing to the
                  eye - 4/15/18
170
171             int MaxQuantity = 299; // max amount of words referenced in array //
                   [0] to [299]
172             int MaxMissesPerSet = 400;
173             //after 400 missed recurive calls, the zone will expand and
                  missesPerSet is reset to 0
174             //increment will go down by one every time until it reaches zero;
175             if (setMisses > MaxMissesPerSet && Increment > 0)
176             {
177                 Increment--;
```

```
178                 setMisses = 0;
179             }
180
181             //while you have not reached the last word asked for...keep placing words
182             if (ArrayPlace < MaxQuantity)
183             {
184                 //create an existing point to start with
185                 Point place = new Point();
186                 //border rectangles so words tdo not land on edge, incremental placemen
187                 //keeps this to a minimum already, but it is good to keep it
188                 Rectangle rightBorder = new Rectangle(850, 1, 10, 589);
189                 Rectangle bottomBorder = new Rectangle(1, 580, 859, 589);
190                 //incremental placement algorithm - HERE IS THE MEAT AND POTATOES
191                 //sets placeholder point equal to the random values between the allowed X values
192                 //Defined as between the smallest X placement plus the incremental values
193                 //and
194                 //the largest X placement minus the incremental values
195                 place.X = rnd.Next(MINWIDTH + (Increment * incWidth), MAXWIDTH - (Increment * incWidth));
196                 //sets placeholder point equal to the random values between the allowed Y values
197                 place.Y = rnd.Next(MINHEIGHT + (Increment * incHeight), MAXHEIGHT - (Increment * incHeight));
198                 //sets the location of the word to this new... partly random ... placement
199                 x.Location = place;
200                 //sets text to the current word within the word array
201                 x.Text = WordArray[ArrayPlace];
202                 //alternates colors between every 100
203                 if (ArrayPlace < 100)
204                 {
205                     x.ForeColor = ColorArray[0];
206                 }
207                 else if (ArrayPlace < 200)
208                 {
209                     x.ForeColor = ColorArray[1];
210                 }
211                 else
212                 {
213                     x.ForeColor = ColorArray[2];
214                 }
215                 //Another MEAT AND POTATOES ALGORITHM
216                 //Font name is set to user input, default "Stencil"
217                 //Font size is set to the integer cast of a Log Op. of the actual quantity of the word
218                 //The Log operation makes all words visible and the highly common words not seem GIANT
```

```
219                 //Add 1 to the quantity in case of a log of (1), so not to equal
                       zero
220                 //multiply by constant of 7 (fontMult) for words to be of
                      readable size
221                 //fontMult can change based on inherent changes in sources
222                 x.Font = new Font(MasterFont, (int)(Math.Log((QuantityArray
                      [ArrayPlace] + 1)) * fontMult));
223
224
225                 //Goes through the controls to find the groupBox Control
226                 foreach (Control c1 in this.Controls)
227                 {
228                     if (c1.ToString().StartsWith
                        ("System.Windows.Forms.GroupBox"))
229                     {
230                         //Once groupBox Control is found
231                         //Go through each individual label, by order of tabIndex
232                         foreach (Control c2 in c1.Controls)
233                         {
234                             //checks if this label is being compared to istelf,
                        checks for collision with other labels, the right boarder,
                        and the bottom boarder
235                             if (!c2.Equals(x) && x.Bounds.IntersectsWith
                        (c2.Bounds) || x.Bounds.IntersectsWith(rightBorder) ||
                        x.Bounds.IntersectsWith(bottomBorder))
236                             {
237
238                                 totalMisses++; //count displayed by the
                        misses label after run button is clicked
239                                 setMisses++; //how many collisions the
                        program has in its current level of incrementation
240                                 ArrangeLabel(x); //if there's a collision it
                         keeps trying to place by calling the function recursively
241
242                             }
243
244                         }
245                     }
246                 }
247             }
248         }
249
250         private void ArrangeLabel4Shape(Label x)
251         {//To arrange Label with information across groupbox, in a randomized
              way, inside a given shape
252             int HEARTminX = 100;
253             int HEARTmaxX = 750;
254             int HEARTminY = 30;
255             int HEARTmaxY = 560;
256             int HEARTincX = 20;
257             int HEARTincY = 20;
258
```

```
259             int MaxQuantity = 299;
260             int MaxMissesPerSet = 400;
261
262             Color WHITE = new Color();
263             WHITE = Color.FromArgb(255, 255, 255, 255);
264
265             if (setMisses > MaxMissesPerSet && Increment > 0)
266             {
267                 Increment--;
268                 setMisses = 0;
269             }
270             if (ArrayPlace < MaxQuantity)
271             {
272
273                 Point place = new Point();
274                 Rectangle rightBorder = new Rectangle(HEARTmaxX, 1, HEARTminX,
                      589);
275                 Rectangle bottomBorder = new Rectangle(1, HEARTmaxY, 859,
                      HEARTminY);
276                 Rectangle topBorder = new Rectangle(1, 1, 859, HEARTminY);
277                 Rectangle leftBorder = new Rectangle(1, 1, HEARTminX, 589);
278                 place.X = rnd.Next(HEARTminX + (Increment * HEARTincX),
                      HEARTmaxX - (Increment * HEARTincX));
279                 place.Y = rnd.Next(HEARTminY + (Increment * HEARTincY),
                      HEARTmaxY - (Increment * HEARTincY));
280                 x.Location = place;
281                 x.Text = WordArray[ArrayPlace];
282                 if (ArrayPlace < 100)
283                 {
284                     x.ForeColor = ColorArray[0];
285                 }
286                 else if (ArrayPlace < 200)
287                 {
288                     x.ForeColor = ColorArray[1];
289                 }
290                 else
291                 {
292                     x.ForeColor = ColorArray[2];
293                 }
294                 x.Font = new Font(MasterFont, (int)(Math.Log((QuantityArray
                      [ArrayPlace] + 1)) * 6.25));
295                 x.Anchor = (AnchorStyles.Top | AnchorStyles.Left |
                      AnchorStyles.Bottom | AnchorStyles.Right);
296                 foreach (Control c1 in this.Controls)
297                 {
298                     if (c1.ToString().StartsWith
                      ("System.Windows.Forms.GroupBox"))
299                     {
300                         foreach (Control c2 in c1.Controls)
301                         {
302                             //Color screenColor = GetColorFromScreen(place);
303                             if (!c2.Equals(x) && x.Bounds.IntersectsWith
```

```
                        (c2.Bounds) ||
304                             x.Bounds.IntersectsWith(rightBorder) ||
305                             x.Bounds.IntersectsWith(bottomBorder) ||
306                             x.Bounds.IntersectsWith(leftBorder) ||
307                             x.Bounds.IntersectsWith(topBorder))
308                         {
309
310                             totalMisses++; //count displayed by the misses
                        label after run button is clicked
311                             setMisses++; //how many collisions the program
                        has in its current level of incrementation
312                             ArrangeLabel4Shape(x); //if there's a collision
                        it keeps trying to place by calling the function over and
                        over
313                         }
314
315                     }
316                 }
317             }
318         }
319     }
320     private void RunButton_Click(object sender, EventArgs e)
321     {
322         //The real driver of all funcionality
323         //Makes the cloud appear
324         //Sets the necessary variables, then calls functions to create tag
              cloud
325
326         //counter which is equal to amount of words user asks for
327         decimal counter = 0;
328
329         counter = UserNumWords.Value - 1;  // grabs value from the numerical
               updown in the UI
330                                            //subtract by one for the array
                    to work correctly
331         int refreshStep = 25;//how often the program will ask to condese the
               words
332         int showStep = 50;//how often the program will ask the form to
              update() itself
333         bool isShape = false;//sets the default value to not use a shaped
              background
334
335         for (int i = 0; i <= counter; i++)//for loop which stops after all
              words
336         {
337             if (!ShapeBack)//if no shape detected progress as normal
338             {
339                 ArrangeLabel(GenerateLabel());
340             }
341             else //otherwise progress as if there is a shape
342             {
343                 ArrangeLabel(GenerateLabel());
```

```
344                     isShape = true;//set off shape trigger
345                 }
346 
347                 ArrayPlace++;//continue to next place in arrays
348                 if (ArrayPlace % refreshStep == 0)//condesnse every 25 words
                      placed
349                 {
350                     //Update();
351                     Condense();
352                     if (ArrayPlace % showStep == 0)//update screen every 50
353                         Update();
354                 }
355 
356             }
357             //Special coditions of condense applied when background shape exists
358             if (isShape)//if there is a shaped background
359             {
360                 while (cleanTheShape() > 0)//delete words that are outside the
                      shape
361                 {                           //until there exist no more
362                     cleanTheShape();
363                 }
364             }
365 
366 
367             //makes sure to resent all buttons and constants that are neccessary
368             ArrayPlace = 0; //reset to read starting at 0
369             if (!ShapeBack)
370             {
371                 BtnClean.Enabled = true; //allow user clean up if not using a
                      shape
372             }
373             ClearButton.Enabled = true; //allow wipe of page
374             btnScore.Enabled = true; //allow score to be recorded - not
                  essential
375             lblMisses.Text = totalMisses.ToString(); //display misses/
                  inefficiencies
376             RunButton.Enabled = false; //do not allow for another run until
                  screen is cleared
377             RunButton.BackColor = Color.Maroon; //make apparent by changing run
                  to dark red
378             changeBackgroundToolStripMenuItem.Enabled = false; //user is not
                  allowed to change background shape
379             openSourceToolStripMenuItem.Enabled = false; //user is not allowed
                  to change the source of data
380             //toolTips to notify user why they cannot use background and change
                  source
381             changeBackgroundToolStripMenuItem.ToolTipText = "must clear screen
                  to enable shape modifications";
382             openSourceToolStripMenuItem.ToolTipText = "must clear screen to
                  enable source modifcation";
383         }
```

```
384
385         private void ClearButton_Click(object sender, EventArgs e)
386         {
387             //deletes every label on the screen
388             //resets buttons, enables all UI, and clears scores and misses
389             canvasBox.Controls.Clear();
390             btnScore.Enabled = false;
391             BtnClean.Enabled = false;
392             totalWordQuantity = 0;
393             Increment = 10;
394             lblScore.Text = "Score N/A";
395             lblMisses.Text = "";
396             totalMisses = 0;
397             RunButton.Enabled = true;
398             RunButton.BackColor = Color.PaleGreen; //resets color of RUN to
                  green
399             changeBackgroundToolStripMenuItem.Enabled = true;
400             openSourceToolStripMenuItem.Enabled = true;
401             changeBackgroundToolStripMenuItem.ToolTipText = "";
402             openSourceToolStripMenuItem.ToolTipText = "";
403         }
404         private void UserNumWords_ValueChanged(object sender, EventArgs e)
405         {
406             //makes sure the value of zero cannot be run
407             if (UserNumWords.Value == 0)
408             {
409                 RunButton.Enabled = false;
410             }
411             else
412             {
413                 RunButton.Enabled = true;
414             }
415         }
416
417         private void btnScore_Click(object sender, EventArgs e)
418         {
419             //Need for this function may be unnessesary
420             //Total white space is not a good mark, its only white space between
                    words
421             //that matters....idk how to just count that or if its possible
422             int OffsetOnBoarder = 9; // I made bounding rectangles on the
                  boarder, from X 850 to 859, and Y 589 to 580
423             double usedArea = 0;
424             double groupBoxArea = ((MAXHEIGHT - MINHEIGHT) - OffsetOnBoarder) *
                  ((MAXWIDTH - MINWIDTH) - OffsetOnBoarder);
425             double usedToWhite;
426             foreach (Control c1 in this.Controls)
427             {
428                 if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
429                 {
430                     foreach (Control c2 in c1.Controls)
431                     {
```

```
432                             usedArea += (c2.Size.Height * c2.Size.Width);
433                         }
434                     }
435                 }
436                 usedToWhite = Math.Round(usedArea / groupBoxArea, 6);
437                 lblScore.Text = usedArea + " / " + groupBoxArea + " = " +
                      (usedToWhite * 100) + "%";
438             }
439
440             private void Form1_Load(object sender, EventArgs e)
441             {
442                 //could be useful to load things here
443             }
444             private void ReadInstances(int fileChoice)
445             {
446                 //reads .txt file based off of user choice(int fileChoice)
447                 //creates new streamreader for both the word file and quantity file
448                 //while the count does not exceed 299(MaxWords) keep reading line by
                      line
449                 //into the WordArray[at count] and the QuantityArray[at count]
450                 //identical process for each choice ~~~ could probably be
                      streamlined
451
452                 if (fileChoice == 1)
453                 {
454
455                     System.IO.StreamReader file =
456                     new System.IO.StreamReader(@"NBC2017review.txt");
457                     string line;
458                     int count = 0;
459                     while ((line = file.ReadLine()) != null && count < MaxWords)
460                     {
461                         WordArray[count] = line;
462                         count++;
463                     }
464
465                     string quantity;
466                     System.IO.StreamReader file2 =
467                     new System.IO.StreamReader(@"NBC2017reviewNumbers.txt");
468                     count = 0;
469                     while ((quantity = file2.ReadLine()) != null && count <
                          MaxWords)
470                     {
471                         QuantityArray[count] = Convert.ToInt32(quantity);
472                         totalWordQuantity += QuantityArray[count];
473                         count++;
474                     }
475                 }
476
477                 else if (fileChoice == 2)
478                 {
479                     System.IO.StreamReader file =
```

```
480                 new System.IO.StreamReader(@"LydiaBlogWords.txt");
481                 string line;
482                 int count = 0;
483                 while ((line = file.ReadLine()) != null && count < MaxWords)
484                 {
485                     WordArray[count] = line;
486                     count++;
487                 }
488 
489                 string quantity;
490                 System.IO.StreamReader file2 =
491                 new System.IO.StreamReader(@"LydiaBlogNumbers.txt");
492                 count = 0;
493                 while ((quantity = file2.ReadLine()) != null && count <
                      MaxWords)
494                 {
495                     QuantityArray[count] = Convert.ToInt32(quantity);
496                     totalWordQuantity += QuantityArray[count];
497                     count++;
498                 }
499             }
500             else if (fileChoice == 3)
501             {
502                 System.IO.StreamReader file =
503                 new System.IO.StreamReader(@"InceptionWords.txt");
504                 string line;
505                 int count = 0;
506                 while ((line = file.ReadLine()) != null && count < MaxWords)
507                 {
508                     WordArray[count] = line;
509                     count++;
510                 }
511 
512                 string quantity;
513                 System.IO.StreamReader file2 =
514                 new System.IO.StreamReader(@"InceptionNumbers.txt");
515                 count = 0;
516                 while ((quantity = file2.ReadLine()) != null && count <
                      MaxWords)
517                 {
518                     QuantityArray[count] = Convert.ToInt32(quantity);
519                     totalWordQuantity += QuantityArray[count];
520                     count++;
521                 }
522             }
523 
524             else if (fileChoice == 4)
525             {
526                 System.IO.StreamReader file =
527                 new System.IO.StreamReader(@"StNorbertWords.txt");
528                 string line;
529                 int count = 0;
```

```
530                 while ((line = file.ReadLine()) != null && count < MaxWords)
531                 {
532                     WordArray[count] = line;
533                     count++;
534                 }
535 
536                 string quantity;
537                 System.IO.StreamReader file2 =
538                 new System.IO.StreamReader(@"StNorbertNumbers.txt");
539                 count = 0;
540                 while ((quantity = file2.ReadLine()) != null && count <
                      MaxWords)
541                 {
542                     QuantityArray[count] = Convert.ToInt32(quantity);
543                     totalWordQuantity += QuantityArray[count];
544                     count++;
545                 }
546             }
547 
548             else if (fileChoice == 5)
549             {
550                 System.IO.StreamReader file =
551                 new System.IO.StreamReader(@"ComputerScienceWords.txt");
552                 string line;
553                 int count = 0;
554                 while ((line = file.ReadLine()) != null && count < MaxWords)
555                 {
556                     WordArray[count] = line;
557                     count++;
558                 }
559 
560                 string quantity;
561                 System.IO.StreamReader file2 =
562                 new System.IO.StreamReader(@"ComputerScienceNumbers.txt");
563                 count = 0;
564                 while ((quantity = file2.ReadLine()) != null && count <
                      MaxWords)
565                 {
566                     QuantityArray[count] = Convert.ToInt32(quantity);
567                     totalWordQuantity += QuantityArray[count];
568                     count++;
569                 }
570             }
571 
572             else if (fileChoice == 6)
573             {
574                 System.IO.StreamReader file =
575                 new System.IO.StreamReader(@"ArtistStatementWords.txt");
576                 string line;
577                 int count = 0;
578                 while ((line = file.ReadLine()) != null && count < MaxWords)
579                 {
```

```
580                     WordArray[count] = line;
581                     count++;
582                 }
583
584                 string quantity;
585                 System.IO.StreamReader file2 =
586                 new System.IO.StreamReader(@"ArtistStatementNumbers.txt");
587                 count = 0;
588                 while ((quantity = file2.ReadLine()) != null && count <
                      MaxWords)
589                 {
590                     QuantityArray[count] = Convert.ToInt32(quantity);
591                     totalWordQuantity += QuantityArray[count];
592                     count++;
593                 }
594             }
595         }
596
597         private void Condense()
598         {
599             //Function that moves labels closer together towards the middle
600             //while keeping in mind zero collision and highest priority on
                  whitespace
601             //This function mainly makes Hypothetical calls to IsCollision with
                  given label
602
603             // McVey Has Helped me find the bug of Condense()!!!! Not checking
                  for self - 4/20/18
604
605             //offset for large words, and small words respectively
606             //large words can only move small offsets, small can move much
                  farther
607             //techincally can check by 1, but takes a toll on time
608             //int OFFSETtiny = 1;
609             int OFFSETsmall = 5;
610             int OFFSETmed = 20;
611             int OFFSETlarge = 20;
612             int OFFSETlargest = 40;
613             int OFFSETHUGE = 80;
614
615             //marks middle of groupBox
616             int MiddleX = 360;
617             int MiddleY = 280;
618             //850/2 = 425
619             //580/2 = 290
620             //since the above coordinates would be applied to the top left of
                  each label
621             //I changed the above numbers to be more centered.
622             //Considering that detail reflected in a more centered tagCloud
623
624             //SAME FOREACH LOOP TO REACH EACH LABEL ~~~ could possibly refine
                  but works as needed
```

```
625
626             foreach (Control c1 in this.Controls)
627             {
628                 if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
629                 {
630                     foreach (Control c2 in c1.Controls)
631                     {
632                         Point loc = c2.Location;//point to hold dummy numbers
                        until if statement is actually true
633                         //MessageBox.Show(c2.Text);   DO NOT DO WITH HIGH
                        NUMBERS! // for debug only
634                         //If the X location is to the right of the middle, see
                        how far it can move left without collision
635                         //try with largest values first, doing lazy evaluation
636                         if (c2.Location.X > MiddleX)
637                         {
638                             if (!IsCollision(loc.X - OFFSETHUGE, loc.Y,
                            c2.Width, c2.Height, (Label)c2))
639                             {
640                                 loc.X = loc.X - OFFSETHUGE;
641                                 c2.Location = loc;
642                             }
643                             else if (!IsCollision(loc.X - OFFSETlargest, loc.Y,
                            c2.Width, c2.Height, (Label)c2))
644                             {
645                                 loc.X = loc.X - OFFSETlargest;
646                                 c2.Location = loc;
647                             }
648                             else if (!IsCollision(loc.X - OFFSETlarge, loc.Y,
                            c2.Width, c2.Height, (Label)c2))
649                             {
650                                 loc.X = loc.X - OFFSETlarge;
651                                 c2.Location = loc;
652                             }
653                             else if (!IsCollision(loc.X - OFFSETmed, loc.Y,
                            c2.Width, c2.Height, (Label)c2))
654                             {
655                                 loc.X = loc.X - OFFSETmed;
656                                 c2.Location = loc;
657                             }
658                             else if (!IsCollision(loc.X - OFFSETsmall, loc.Y,
                            c2.Width, c2.Height, (Label)c2))
659                             {
660                                 loc.X = loc.X - OFFSETsmall;
661                                 c2.Location = loc;
662                             }
663                             //else if (!IsCollision(loc.X - OFFSETtiny, loc.Y,
                            c2.Width, c2.Height, (Label)c2))
664                             //{
665                             //    loc.X = loc.X - OFFSETtiny;
666                             //    c2.Location = loc;
667                             //}
```

```
668                         }
669                         else
670                         {   //The X location must be to the left of the middle, see how far it can move right without collision
671                             //try with largest values first, doing lazy evaluation
672                             if (!IsCollision(loc.X + OFFSETHUGE, loc.Y, c2.Width, c2.Height, (Label)c2))
673                             {
674                                 loc.X = loc.X + OFFSETHUGE;
675                                 c2.Location = loc;
676                             }
677                             else if (!IsCollision(loc.X + OFFSETlargest, loc.Y , c2.Width, c2.Height, (Label)c2))
678                             {
679                                 loc.X = loc.X + OFFSETlargest;
680                                 c2.Location = loc;
681                             }
682                             else if (!IsCollision(loc.X + OFFSETlarge, loc.Y, c2.Width, c2.Height, (Label)c2))
683                             {
684                                 loc.X = loc.X + OFFSETlarge;
685                                 c2.Location = loc;
686                             }
687                             else if (!IsCollision(loc.X + OFFSETmed, loc.Y, c2.Width, c2.Height, (Label)c2))
688                             {
689                                 loc.X = loc.X + OFFSETmed;
690                                 c2.Location = loc;
691                             }
692                             else if (!IsCollision(loc.X + OFFSETsmall, loc.Y, c2.Width, c2.Height, (Label)c2))
693                             {
694                                 loc.X = loc.X + OFFSETsmall;
695                                 c2.Location = loc;
696                             }
697                             //else if (!IsCollision(loc.X + OFFSETtiny, loc.Y, c2.Width, c2.Height, (Label)c2))
698                             //{
699                             //    loc.X = loc.X + OFFSETtiny;
700                             //    c2.Location = loc;
701                             //}
702                         }
703                         if (c2.Location.Y > MiddleY)
704                         {
705                             //If the Y location is Below the middle, see how far it can move up without collision
706                             //try with largest values first, doing lazy evaluation
707                             if (!IsCollision(loc.X , loc.Y - OFFSETHUGE, c2.Width, c2.Height, (Label)c2))
708                             {
```

```
709                                 loc.Y = loc.Y - OFFSETHUGE;
710                                 c2.Location = loc;
711                             }
712                             else if (!IsCollision(loc.X, loc.Y - OFFSETlargest,
                        c2.Width, c2.Height, (Label)c2))
713                             {
714                                 loc.Y = loc.Y - OFFSETlargest;
715                                 c2.Location = loc;
716                             }
717                             else if (!IsCollision(loc.X, loc.Y - OFFSETlarge,
                        c2.Width, c2.Height, (Label)c2))
718                             {
719                                 loc.Y = loc.Y - OFFSETlarge;
720                                 c2.Location = loc;
721                             }
722                             else if (!IsCollision(loc.X, loc.Y - OFFSETmed,
                        c2.Width, c2.Height, (Label)c2))
723                             {
724                                 loc.Y = loc.Y - OFFSETmed;
725                                 c2.Location = loc;
726                             }
727                             else if (!IsCollision(loc.X, loc.Y - OFFSETsmall,
                        c2.Width, c2.Height, (Label)c2))
728                             {
729                                 loc.Y = loc.Y - OFFSETsmall;
730                                 c2.Location = loc;
731                             }
732                             //else if (!IsCollision(loc.X , loc.Y - OFFSETtiny,
                        c2.Width, c2.Height, (Label)c2))
733                             //{
734                             //    loc.Y = loc.Y - OFFSETtiny;
735                             //    c2.Location = loc;
736                             //}
737                         }
738                         else
739                         {
740                             //The Y location must be above the middle, see how
                        far it can move down without collision
741                             //try with largest values first, doing lazy
                        evaluation
742                             if (!IsCollision(loc.X, loc.Y + OFFSETHUGE,
                        c2.Width, c2.Height, (Label)c2))
743                             {
744                                 loc.Y = loc.Y + OFFSETHUGE;
745                                 c2.Location = loc;
746                             }
747                             else if (!IsCollision(loc.X, loc.Y + OFFSETlargest,
                        c2.Width, c2.Height, (Label)c2))
748                             {
749                                 loc.Y = loc.Y + OFFSETlargest;
750                                 c2.Location = loc;
751                             }
```

```
752                             else if (!IsCollision(loc.X, loc.Y + OFFSETlarge,
                        c2.Width, c2.Height, (Label)c2))
753                             {
754                                 loc.Y = loc.Y + OFFSETlarge;
755                                 c2.Location = loc;
756                             }
757                             else if (!IsCollision(loc.X, loc.Y + OFFSETmed,
                        c2.Width, c2.Height, (Label)c2))
758                             {
759                                 loc.Y = loc.Y + OFFSETmed;
760                                 c2.Location = loc;
761                             }
762                             else if (!IsCollision(loc.X, loc.Y + OFFSETsmall,
                        c2.Width, c2.Height, (Label)c2))
763                             {
764                                 loc.Y = loc.Y + OFFSETsmall;
765                                 c2.Location = loc;
766                             }
767                             //else if (!IsCollision(loc.X, loc.Y + OFFSETtiny,
                        c2.Width, c2.Height, (Label)c2))
768                             //{
769                             //    loc.Y = loc.Y + OFFSETtiny;
770                             //    c2.Location = loc;
771                             //}
772                         }
773                         Update(); //update after movement
774                         //otherwise the cloud looks much sloppier
775                     }
776                 }
777             }
778         }
779
780         private void BtnClean_Click(object sender, EventArgs e)
781         {
782            //made this just a call to one main function
783            //allow user to condense as they desire, if not using a shape
784            Condense();
785         }
786
787         private Boolean IsCollision(int LocationX, int LocationY, int sizeX, int
               sizeY, Label lbl)
788         {
789             //Collision function - Checks to see if given label with new
                  coordinates will
790             //collide with an existing label. If this function returns true, it
                  means that
791             //it collided with an existing label
792             //otherwise it will return false, resulting in a placed label
793
794             //creates rectangle that stands in place of the label
795             //for efficiency and allocation reasons - do not want another label
                  to be drawn/moved
```

```
796             Rectangle fromGiven = new Rectangle(LocationX, LocationY, sizeX,
                  sizeY);
797
798             foreach (Control c1 in this.Controls)
799             {
800                 if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
801                 {
802                     foreach (Control c2 in c1.Controls)
803                     {
804                         if(!c2.Equals(lbl)) //For the longest time I did not
                          check for collision of itself ;(
805                                             //Thank you McVey
806                         if (c2.Bounds.IntersectsWith(fromGiven))
807                         {
808                             return true;
809                         }
810                     }
811                 }
812             }
813             return false;
814         }
815
816         private void radBtnBackColor_CheckedChanged(object sender, EventArgs e)
817         {
818             //debug feature to display how much space is wasted with label
                  controls
819             if (radBtnBackColor.Checked == true)
820             {
821                 foreach (Control c1 in this.Controls)
822                 {
823                     if (c1.ToString().StartsWith
                      ("System.Windows.Forms.GroupBox"))
824                     {
825                         foreach (Control c2 in c1.Controls)
826                         {
827                             c2.BackColor = Color.DimGray;
828                         }
829                     }
830                 }
831             }
832             Update();
833         }
834
835         private void radBackColorOff_CheckedChanged(object sender, EventArgs e)
836         {//does not work correctly
837             if (radBtnBackColor.Checked == false)
838             {
839                 foreach (Control c1 in this.Controls)
840                 {
841                     if (c1.ToString().StartsWith
                      ("System.Windows.Forms.Groupbox"))
842                     {
```

```
843                         foreach (Control c2 in c1.Controls)
844                         {
845                             c2.BackColor = Color.White;
846                         }
847                     }
848                 }
849             }
850             Update();
851         }
852
853         private void saveCloudAsToolStripMenuItem_Click(object sender, EventArgs
              e)
854         {
855             //saves a screenshot of the groupbox control, which contains the
                wordTagCloud
856             //Creates a bitmap of the groupbox, based off the client rectangle
857             //Has sometimes saved a snapshot with the saveDialog box still in
                the frame
858             //^^not good
859
860             Bitmap bitmap = new Bitmap(canvasBox.Width, canvasBox.Height);
861             SaveFileDialog sfd = new SaveFileDialog();
862             sfd.InitialDirectory = "C:\\Users\\admin\\Desktop\\CSCICapstone2.0.1
                \\CSCICapstone2.0.1\\bin\\Debug";
863             sfd.Filter = "Images|*.png;*.bmp;*.jpg";
864             ImageFormat format = ImageFormat.Png;
865             if (sfd.ShowDialog() == System.Windows.Forms.DialogResult.OK)
866             {
867                 string ext = System.IO.Path.GetExtension(sfd.FileName);
868                 switch (ext)
869                 {
870                     case ".jpg":
871                         format = ImageFormat.Jpeg;
872                         break;
873                     case ".bmp":
874                         format = ImageFormat.Bmp;
875                         break;
876                 }
877
878                 Graphics g = Graphics.FromImage(bitmap);
879                 Rectangle rect = canvasBox.RectangleToScreen
                    (canvasBox.ClientRectangle);
880                 rect.Y = rect.Y + 5;
881                 g.CopyFromScreen(rect.Location, Point.Empty, canvasBox.Size);
882             }
883             else
884             {
885                 return;
886             }
887             bitmap.Save(sfd.FileName, format);
888         }
889
```

```
890         public Color GetColorFromScreen (Point P)
891         {   //Function for finding the pixel color at a given point
892             //used to check if a label is in/out of shape
893
894
895             Rectangle rect = new Rectangle(P, new Size(2, 2));
896
897             Color grab = map.GetPixel(P.X, P.Y);
898             return grab;
899         }
900
901
902         private int cleanTheShape()
903         {
904             //Function deletes labels that exist outside of the shape
905             //uses getColorFromScreen to determine if label needs to be deleted
906             //if color behind label is white - keep it!
907             //if color behind label is default Control Color - delete it!
908             //if deletes a label, the counter outOfShape goes up by one
909             //return outOfShape
910
911             int outOfShape = 0;
912             Color WHITE = new Color();
913             WHITE = Color.FromArgb(255, 255, 255, 255);
914             Point middle = new Point();
915             Point middleTop = new Point();
916
917             foreach (Control c1 in this.Controls)
918             {
919                 if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
920                 {
921                     foreach (Control c2 in c1.Controls)
922                     {
923                         //check color at the top left of label
924                         Color Screengrab = GetColorFromScreen(c2.Location);
925                         if (Screengrab != WHITE)
926                         {
927                             //MessageBox.Show("should delete" + c2.Text);
928                             c2.Dispose();
929                             outOfShape++;
930                         }
931                         //check at the middle of the label
932                         middle.X = c2.Location.X + (int) (.5 * c2.Width);
933                         middle.Y = c2.Location.Y + (int) (.5 * c2.Height);
934                         Screengrab = GetColorFromScreen(middle);
935                         if (Screengrab != WHITE)
936                         {
937                             //MessageBox.Show("should delete" + c2.Text);
938                             c2.Dispose();
939                             outOfShape++;
940                         }
941                         //check at the middleTop of the label
```

```
942                         middleTop.X = c2.Location.X + (int) (.5 *c2.Width);
943                         middleTop.Y = c2.Location.Y;
944                         Screengrab = GetColorFromScreen(middleTop);
945                         if (Screengrab != WHITE)
946                         {
947                             //MessageBox.Show("should delete" + c2.Text);
948                             c2.Dispose();
949                             outOfShape++;
950                         }
951                     }
952                 }
953             }
954         return outOfShape; //return the number of labels deleted
955         }
956
957         private void heartToolStripMenuItem_Click(object sender, EventArgs e)
958         {
959             //Menu Tool strip for the UI that lets the user pic a background
960             //Picks the heart Background
961             //Disables the "Clean It Up" button
962             //all of these are specific to the shapes I have designated
963
964             original = Image.FromFile("Heart2Control.png");
965             map = new Bitmap(original, canvasBox.Width, canvasBox.Height);
966             canvasBox.BackgroundImage = original;
967             ShapeBack = true;
968             heartToolStripMenuItem.Checked = true;
969
970             blankToolStripMenuItem.Checked = false;
971             arrowToolStripMenuItem.Checked = false;
972             starToolStripMenuItem.Checked = false;
973             lightningBoltToolStripMenuItem.Checked = false;
974             calloutToolStripMenuItem.Checked = false;
975
976             BtnClean.Enabled = false;
977         }
978
979         private void calloutToolStripMenuItem_Click(object sender, EventArgs e)
980         {
981             original = Image.FromFile("WordBubbleControl.png");
982             map = new Bitmap(original, canvasBox.Width, canvasBox.Height);
983             canvasBox.BackgroundImage = original;
984             ShapeBack = true;
985             calloutToolStripMenuItem.Checked = true;
986
987             blankToolStripMenuItem.Checked = false;
988             arrowToolStripMenuItem.Checked = false;
989             starToolStripMenuItem.Checked = false;
990             lightningBoltToolStripMenuItem.Checked = false;
991             heartToolStripMenuItem.Checked = false;
992
993             BtnClean.Enabled = false;
```

```
994         }
995
996         private void lightningBoltToolStripMenuItem_Click(object sender,
              EventArgs e)
997         {
998             original = Image.FromFile("LightningControl1.png");
999             map = new Bitmap(original, canvasBox.Width, canvasBox.Height);
1000            canvasBox.BackgroundImage = original;
1001            ShapeBack = true;
1002            lightningBoltToolStripMenuItem.Checked = true;
1003
1004            blankToolStripMenuItem.Checked = false;
1005            arrowToolStripMenuItem.Checked = false;
1006            starToolStripMenuItem.Checked = false;
1007            calloutToolStripMenuItem.Checked = false;
1008            heartToolStripMenuItem.Checked = false;
1009
1010            BtnClean.Enabled = false;
1011        }
1012
1013        private void starToolStripMenuItem_Click(object sender, EventArgs e)
1014        {
1015            original = Image.FromFile("StarControl.png");
1016            map = new Bitmap(original, canvasBox.Width, canvasBox.Height);
1017            canvasBox.BackgroundImage = original;
1018            ShapeBack = true;
1019            starToolStripMenuItem.Checked = true;
1020
1021            blankToolStripMenuItem.Checked = false;
1022            arrowToolStripMenuItem.Checked = false;
1023            lightningBoltToolStripMenuItem.Checked = false;
1024            calloutToolStripMenuItem.Checked = false;
1025            heartToolStripMenuItem.Checked = false;
1026
1027            BtnClean.Enabled = false;
1028        }
1029        private void arrowToolStripMenuItem_Click(object sender, EventArgs e)
1030        {
1031            original = Image.FromFile("RightArrowControl.png");
1032            map = new Bitmap(original, canvasBox.Width, canvasBox.Height);
1033            canvasBox.BackgroundImage = original;
1034            ShapeBack = true;
1035            arrowToolStripMenuItem.Checked = true;
1036
1037            blankToolStripMenuItem.Checked = false;
1038            starToolStripMenuItem.Checked = false;
1039            lightningBoltToolStripMenuItem.Checked = false;
1040            calloutToolStripMenuItem.Checked = false;
1041            heartToolStripMenuItem.Checked = false;
1042
1043            BtnClean.Enabled = false;
1044        }
```

```
1045
1046        private void blankToolStripMenuItem_Click(object sender, EventArgs e)
1047        {
1048            original = Image.FromFile("Control.png");
1049            map = new Bitmap(original, canvasBox.Width, canvasBox.Height);
1050            canvasBox.BackgroundImage = original;
1051            ShapeBack = false;
1052            blankToolStripMenuItem.Checked = true;
1053
1054            arrowToolStripMenuItem.Checked = false;
1055            starToolStripMenuItem.Checked = false;
1056            lightningBoltToolStripMenuItem.Checked = false;
1057            calloutToolStripMenuItem.Checked = false;
1058            heartToolStripMenuItem.Checked = false;
1059
1060            BtnClean.Enabled = false;
1061        }
1062
1063        private void stencilToolStripMenuItem_Click(object sender, EventArgs e)
1064        {
1065            //Sets the font to the choosen font - Re-runs the whole program with
                    new font
1066            MasterFont = FontArray[0];
1067
1068            foreach (Control c1 in this.Controls)
1069            {
1070                if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
1071                {
1072                    foreach (Control c2 in c1.Controls)
1073                    {
1074                        c2.Font = new Font(MasterFont, c2.Font.Size);
1075                    }
1076                }
1077            }
1078
1079            timesNewRomanToolStripMenuItem.Checked = false;
1080            microsoftSansSerifToolStripMenuItem.Checked = false;
1081            wingdingsToolStripMenuItem.Checked = false;
1082            impactToolStripMenuItem.Checked = false;
1083            harlowSolidToolStripMenuItem.Checked = false;
1084            cooperToolStripMenuItem.Checked = false;
1085            comicSansMSToolStripMenuItem.Checked = false;
1086
1087            stencilToolStripMenuItem.Checked = true;
1088
1089            ClearButton_Click(sender, e);
1090            RunButton_Click(sender, e);
1091
1092            //Condense();
1093        }
1094
1095        private void timesNewRomanToolStripMenuItem_Click(object sender,
```

```
            EventArgs e)
1096        {
1097            MasterFont = FontArray[1];
1098
1099            foreach (Control c1 in this.Controls)
1100            {
1101                if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
1102                {
1103                    foreach (Control c2 in c1.Controls)
1104                    {
1105                        c2.Font = new Font(MasterFont, c2.Font.Size);
1106                    }
1107                }
1108            }
1109
1110            stencilToolStripMenuItem.Checked = false;
1111            microsoftSansSerifToolStripMenuItem.Checked = false;
1112            wingdingsToolStripMenuItem.Checked = false;
1113            impactToolStripMenuItem.Checked = false;
1114            harlowSolidToolStripMenuItem.Checked = false;
1115            cooperToolStripMenuItem.Checked = false;
1116            comicSansMSToolStripMenuItem.Checked = false;
1117
1118            timesNewRomanToolStripMenuItem.Checked = true;
1119
1120            ClearButton_Click(sender, e);
1121            RunButton_Click(sender, e);
1122
1123            //Condense();
1124        }
1125
1126        private void microsoftSansSerifToolStripMenuItem_Click(object sender,
            EventArgs e)
1127        {
1128            MasterFont = FontArray[2];
1129
1130            foreach (Control c1 in this.Controls)
1131            {
1132                if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
1133                {
1134                    foreach (Control c2 in c1.Controls)
1135                    {
1136                        c2.Font = new Font(MasterFont, c2.Font.Size);
1137                    }
1138                }
1139            }
1140
1141            stencilToolStripMenuItem.Checked = false;
1142            timesNewRomanToolStripMenuItem.Checked = false;
1143            wingdingsToolStripMenuItem.Checked = false;
1144            impactToolStripMenuItem.Checked = false;
1145            harlowSolidToolStripMenuItem.Checked = false;
```

```
1146            cooperToolStripMenuItem.Checked = false;
1147            comicSansMSToolStripMenuItem.Checked = false;
1148
1149            microsoftSansSerifToolStripMenuItem.Checked = true;
1150
1151            ClearButton_Click(sender, e);
1152            RunButton_Click(sender, e);
1153
1154            //Condense();
1155        }
1156
1157        private void wingdingsToolStripMenuItem_Click(object sender, EventArgs
              e)
1158        {
1159            MasterFont = FontArray[3];
1160
1161            foreach (Control c1 in this.Controls)
1162            {
1163                if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
1164                {
1165                    foreach (Control c2 in c1.Controls)
1166                    {
1167                        c2.Font = new Font(MasterFont, c2.Font.Size);
1168                    }
1169                }
1170            }
1171
1172            stencilToolStripMenuItem.Checked = false;
1173            timesNewRomanToolStripMenuItem.Checked = false;
1174            microsoftSansSerifToolStripMenuItem.Checked = false;
1175            impactToolStripMenuItem.Checked = false;
1176            harlowSolidToolStripMenuItem.Checked = false;
1177            cooperToolStripMenuItem.Checked = false;
1178            comicSansMSToolStripMenuItem.Checked = false;
1179
1180            wingdingsToolStripMenuItem.Checked = true;
1181
1182            ClearButton_Click(sender, e);
1183            RunButton_Click(sender, e);
1184
1185            //Condense();
1186        }
1187
1188        private void impactToolStripMenuItem_Click(object sender, EventArgs e)
1189        {
1190            MasterFont = FontArray[4];
1191
1192            foreach (Control c1 in this.Controls)
1193            {
1194                if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
1195                {
1196                    foreach (Control c2 in c1.Controls)
```

```
                    {
                        c2.Font = new Font(MasterFont, c2.Font.Size);
                    }
                }
            }

            stencilToolStripMenuItem.Checked = false;
            timesNewRomanToolStripMenuItem.Checked = false;
            microsoftSansSerifToolStripMenuItem.Checked = false;
            wingdingsToolStripMenuItem.Checked = false;
            harlowSolidToolStripMenuItem.Checked = false;
            cooperToolStripMenuItem.Checked = false;
            comicSansMSToolStripMenuItem.Checked = false;

            impactToolStripMenuItem.Checked = true;

            ClearButton_Click(sender, e);
            RunButton_Click(sender, e);

            //Condense();
        }

        private void harlowSolidToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MasterFont = FontArray[5];

            foreach (Control c1 in this.Controls)
            {
                if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
                {
                    foreach (Control c2 in c1.Controls)
                    {
                        c2.Font = new Font(MasterFont, c2.Font.Size);
                    }
                }
            }

            stencilToolStripMenuItem.Checked = false;
            timesNewRomanToolStripMenuItem.Checked = false;
            microsoftSansSerifToolStripMenuItem.Checked = false;
            wingdingsToolStripMenuItem.Checked = false;
            impactToolStripMenuItem.Checked = false;
            cooperToolStripMenuItem.Checked = false;
            comicSansMSToolStripMenuItem.Checked = false;

            harlowSolidToolStripMenuItem.Checked = true;

            ClearButton_Click(sender, e);
            RunButton_Click(sender, e);

            //Condense();
```

```
1248        }
1249
1250        private void cooperToolStripMenuItem_Click(object sender, EventArgs e)
1251        {
1252            MasterFont = FontArray[6];
1253
1254            foreach (Control c1 in this.Controls)
1255            {
1256                if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
1257                {
1258                    foreach (Control c2 in c1.Controls)
1259                    {
1260                        c2.Font = new Font(MasterFont, c2.Font.Size);
1261                    }
1262                }
1263            }
1264
1265            stencilToolStripMenuItem.Checked = false;
1266            timesNewRomanToolStripMenuItem.Checked = false;
1267            microsoftSansSerifToolStripMenuItem.Checked = false;
1268            wingdingsToolStripMenuItem.Checked = false;
1269            impactToolStripMenuItem.Checked = false;
1270            harlowSolidToolStripMenuItem.Checked = false;
1271            comicSansMSToolStripMenuItem.Checked = false;
1272
1273            cooperToolStripMenuItem.Checked = true;
1274
1275            ClearButton_Click(sender, e);
1276            RunButton_Click(sender, e);
1277
1278            //Condense();
1279        }
1280
1281        private void comicSansMSToolStripMenuItem_Click(object sender, EventArgs ↵
               e)
1282        {
1283            MasterFont = FontArray[7];
1284
1285            foreach (Control c1 in this.Controls)
1286            {
1287                if (c1.ToString().StartsWith("System.Windows.Forms.GroupBox"))
1288                {
1289                    foreach (Control c2 in c1.Controls)
1290                    {
1291                        c2.Font = new Font(MasterFont, c2.Font.Size);
1292                    }
1293                }
1294            }
1295
1296            stencilToolStripMenuItem.Checked = false;
1297            timesNewRomanToolStripMenuItem.Checked = false;
1298            microsoftSansSerifToolStripMenuItem.Checked = false;
```

```
1299            wingdingsToolStripMenuItem.Checked = false;
1300            impactToolStripMenuItem.Checked = false;
1301            harlowSolidToolStripMenuItem.Checked = false;
1302            cooperToolStripMenuItem.Checked = false;
1303
1304            comicSansMSToolStripMenuItem.Checked = true;
1305
1306            ClearButton_Click(sender, e);
1307            RunButton_Click(sender, e);
1308
1309            //Condense();
1310        }
1311
1312        private void nBC2017InReviewToolStripMenuItem_Click(object sender,
              EventArgs e)
1313        {
1314            //Sets the source data of the tag cloud generator to what the user
                chooses
1315            //default value is 1 for the NBC2017YearInReview article
1316
1317            SourceChoice = 1;
1318            ReadInstances(SourceChoice);
1319
1320            spainBlogToolStripMenuItem.Checked = false;
1321            inceptionExplainedToolStripMenuItem.Checked = false;
1322            stNorbertWikiToolStripMenuItem.Checked = false;
1323            compSciWikiToolStripMenuItem.Checked = false;
1324            artistsStatementsToolStripMenuItem.Checked = false;
1325
1326            nBC2017InReviewToolStripMenuItem.Checked = true;
1327            fontMult = 7;
1328        }
1329
1330        private void spainBlogToolStripMenuItem_Click(object sender, EventArgs
              e)
1331        {
1332            SourceChoice = 2;
1333            ReadInstances(SourceChoice);
1334
1335            inceptionExplainedToolStripMenuItem.Checked = false;
1336            stNorbertWikiToolStripMenuItem.Checked = false;
1337            compSciWikiToolStripMenuItem.Checked = false;
1338            artistsStatementsToolStripMenuItem.Checked = false;
1339            nBC2017InReviewToolStripMenuItem.Checked = false;
1340
1341            spainBlogToolStripMenuItem.Checked = true;
1342            fontMult = 7;
1343        }
1344
1345        private void inceptionExplainedToolStripMenuItem_Click(object sender,
              EventArgs e)
1346        {
```

```
1347                SourceChoice = 3;
1348                ReadInstances(SourceChoice);
1349
1350                stNorbertWikiToolStripMenuItem.Checked = false;
1351                compSciWikiToolStripMenuItem.Checked = false;
1352                artistsStatementsToolStripMenuItem.Checked = false;
1353                nBC2017InReviewToolStripMenuItem.Checked = false;
1354                spainBlogToolStripMenuItem.Checked = false;
1355
1356                inceptionExplainedToolStripMenuItem.Checked = true;
1357                fontMult = 8;
1358            }
1359
1360            private void stNorbertWikiToolStripMenuItem_Click(object sender,
                  EventArgs e)
1361            {
1362                SourceChoice = 4;
1363                ReadInstances(SourceChoice);
1364
1365                inceptionExplainedToolStripMenuItem.Checked = false;
1366                compSciWikiToolStripMenuItem.Checked = false;
1367                artistsStatementsToolStripMenuItem.Checked = false;
1368                nBC2017InReviewToolStripMenuItem.Checked = false;
1369                spainBlogToolStripMenuItem.Checked = false;
1370
1371                stNorbertWikiToolStripMenuItem.Checked = true;
1372                fontMult = 7;
1373            }
1374
1375            private void compSciWikiToolStripMenuItem_Click(object sender, EventArgs
                   e)
1376            {
1377                SourceChoice = 5;
1378                ReadInstances(SourceChoice);
1379
1380                inceptionExplainedToolStripMenuItem.Checked = false;
1381                stNorbertWikiToolStripMenuItem.Checked = false;
1382                artistsStatementsToolStripMenuItem.Checked = false;
1383                nBC2017InReviewToolStripMenuItem.Checked = false;
1384                spainBlogToolStripMenuItem.Checked = false;
1385
1386                compSciWikiToolStripMenuItem.Checked = true;
1387                fontMult = 5;
1388            }
1389
1390            private void artistsStatementsToolStripMenuItem_Click(object sender,
                  EventArgs e)
1391            {
1392                SourceChoice = 6;
1393                ReadInstances(SourceChoice);
1394
1395                inceptionExplainedToolStripMenuItem.Checked = false;
```

```
1396                stNorbertWikiToolStripMenuItem.Checked = false;
1397                compSciWikiToolStripMenuItem.Checked = false;
1398                nBC2017InReviewToolStripMenuItem.Checked = false;
1399                spainBlogToolStripMenuItem.Checked = false;
1400
1401                artistsStatementsToolStripMenuItem.Checked = true;
1402                fontMult = 7;
1403            }
1404
1405            private void changeBackgroundToolStripMenuItem_MouseHover(object sender,
                    EventArgs e)
1406            {
1407                //notifies the user why they cannot change the background while
                      there is
1408                //an active tagcloud
1409                ToolTip changeOptions = new ToolTip();
1410                changeOptions.Show(changeBackgroundToolStripMenuItem.ToolTipText,
                      this, Location, 5000);
1411            }
1412
1413            private void openSourceToolStripMenuItem_MouseHover(object sender,
                  EventArgs e)
1414            {
1415                //notifies the user why they cannot change the source while there is

1416                //an active tagcloud
1417                ToolTip openSource = new ToolTip();
1418                openSource.Show(changeBackgroundToolStripMenuItem.ToolTipText, this,
                       Location, 5000);
1419            }
1420
1421            //plans to create one step color changing toolStrip
1422        }
1423    }
1424
1425
1426
```