# Zena's CS251 Project1

For the first project, we were tasked with becoming familiar with the basic tkinter functionalities -- creating a canvas, making points, using buttons, drop-down menus, and popup windows, etc. In this specific program, we had to make a window with a canvas and a sidebar. The sidebar widgets could be used to choose a color (which changes the color of dots made in the future), change all currently-drawn dots' color, change distribution on both axes (either gaussian or uniform randomness), and actually draw the points. I also did the extensions to change the amount of points drawn, change the shapes, and keep the current distributions (and also shapes!) highlighted and set in the window for their selection.

Task 1 was to make both a keyboard shortcut and a file menu option to clear the canvas of all points. To accomplish this, I made a method in DisplayApp that reset the objects list to an empty one and called canvas.delete("all"). In the buildMenus method, I added in new text and a new command for the menutext and menucmd lists, based on how quit was done.

Task 2 was to add the ability to create a random-colored point wherever you right click, and to be able to left click and drag to move around the canvas. First, in the handleMouseButton1 method, I changed the baseClick field to store the current location tuple, and the handleMouseButton1Motion method moves the shapes according to the difference in the original click location and where the mouse is now. The handleMouseButton2 method just calls for a randomly generated rgb value to be given to a new circle you can create when clicking, and the new point is added to the objects list.

Task 3 was to make a popup window that has options for random distribution in the x and y axes. Using an outline the task linked to, I made two listboxes in the body method, and packed them into the window vertically. Then, I inserted "uniform" and "gaussian" as selectable options in both. In the apply method, which hitting "ok" triggers, I changed the fields for the current x and y distribution options to match the currently active selection in their respective listboxes. I passed in the displayApp class for this reason. Then, when creating the points, I check the strings associated with the x and y distribution fields, and use a different random function depending on their values. This is shown here:

```
    # select currently active indices
    # according to current distribution

    # default active indices
    activeX = 0
    activeY = 0
    if self.displayApp.xDist == "gaussian":
        activeX = 1
    if self.displayApp.yDist == "gaussian":
        activeY = 1
    self.listbox1.activate(activeX)
    self.listbox1.select_set(activeX)

    self.listbox2.activate(activeY)
    self.listbox2.select_set(activeY)
```

For my first extension, making the current distribution settings saved as the defaults when opening the window to change them, I simply used the activate method in the body method of the DistSelection class. I passed in the currently selected index by checking the fields of the displayApp class. I also used the select_set method of the listboxes in the same way to make the currently active distribution highlighted for clarity.

The second extension I did was adding a slider for changing the amount of points to draw. This was a simple matter of adding two lines to buildControls, shown below, and then telling the point-creation method to make points in the range of self.slider.get().

```
    self.slider = tk.Scale(rightcntlframe, from_=1, to=1000, orient = tk.HORIZONTAL)
    self.slider.pack(side = tk.TOP)
```

For the last extension, I added the ability to make square and diamond shapes. I modeled this after the popup window for choosing distribution type, only with one listbox. I had a field for the shape in displayApp, and I checked this when drawing the points. The challenges came with making a diamond. This, unlike a square, cannot be treated similarly to a circle, with two x and two y values. It needs to have 4 of each, and is a

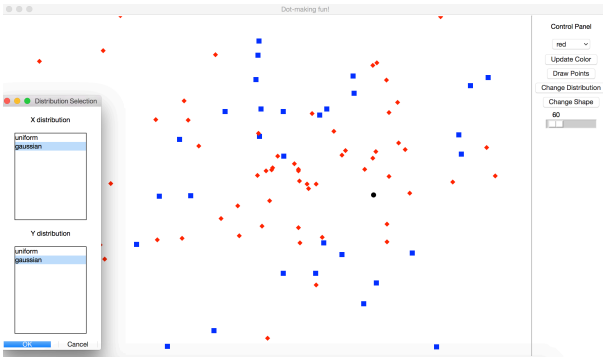
polygon. This also means it needs to be moved differently in the mouse movement-handling method. After a while, I found that I could give shapes I draw tags, and by checking if the shapes had tags, I could differentiate between diamonds and non-diamonds. For diamonds, I moved 8 points, while for others, it was only 4. This is shown below:

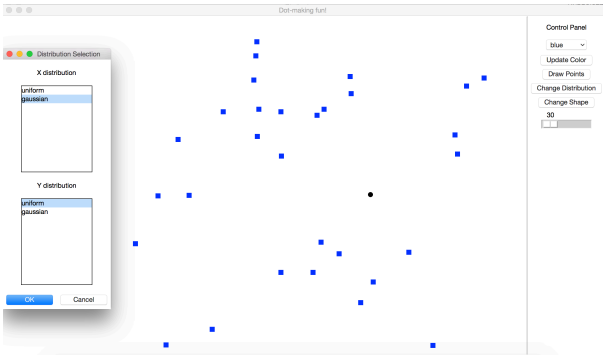
```
# update polygons (diamonds) differently!!
if len(self.canvas.gettags(obj)) != 0:
    #print "POLYGON!!!"
    self.canvas.coords( obj,
                loc[0] + diff[0],
                loc[1] + diff[1],
                loc[2] + diff[0],
                loc[3] + diff[1],
                loc[4] + diff[0],
                loc[5] + diff[1],
                loc[6] + diff[0],
                loc[7] + diff[1],
                 )
else:
    self.canvas.coords( obj,
                loc[0] + diff[0],
                loc[1] + diff[1],
                loc[2] + diff[0],
                loc[3] + diff[1] )
```

And this is where I decided what to draw when placing points:

```
print randX,randY
radius = 5
if self.currShape == "circle":
    pt = self.canvas.create_oval( randX-radius, randY-radius, randX+radius, randY+radius,
                                  fill=self.colorOption.get(), outline='' )
elif self.currShape == "square":
    pt = self.canvas.create_rectangle( randX-radius, randY-radius, randX+radius, randY+radius,
                                       fill=self.colorOption.get(), outline='' )
elif self.currShape == "diamond":
    pt = self.canvas.create_polygon( randX-radius, randY, randX, randY-radius, randX+radius, randY,
                                     randX, randY+radius, # last, connecting line is implied
                                fill=self.colorOption.get(), outline='', tags="POLYGON" )
```

Below are two pictures of my program's functionality. The blue is Gaussian in its x axis, is more sparse than the red, and is is square. Meanwhile, there is a lot of red, which takes diamond form, and has gaussian distribution in both axes. The effect of Gaussian distribution can be seen in the center-clumping we can see in the red. The distribution makes it more likely for points to be near the center, and less likely to be farther away, based on the standard deviation provided. The blue, which has Gaussian distribution on its x axis, shows this trend horizontally, but vertically, one can see that it does not go down into the middle of the y axis very much.




This project familiarized me with important tkinter features, and I was able to problem-solve when doing the extensions. I learned how to trigger popup windows, read, use, and display selection feedback, make buttons, manipulate different shapes on the canvas, and how to use the gaussian random method, for example. It wasn't that difficult getting back into python because of Jan Plan, and I appreciated how I could "pack" widgets into the window instead of manually placing everything like in pygame. Tkinter seems more user-friendly.