First – The Field of Conflict...

Exam 1 is an on line exam. You have the entire class time to do the exam. If you finish early, you may leave or stay and work. If you do not finish by 1:20, you will not receive extra time.

The exam consists of four problems in which you write code. You need to create the folders and files required. You must submit them to the handin folder created for you in the course directory.

The problems focus on the fundamentals of programming with Processing that were presented in homeworks 1-6.

You may access the Processing API. You MAY NOT access any other material.

We will, where possible help you with syntax errors but we will not help you with logic problems.

Non-working code can receive partial credit but it must compile.

Second – The Rules of Engagement…

- **You will write four separate programs for this exam – one program for each question.**
- **Each program must be in a separate folder that has the same name as the .pde file inside it.**
- **You will put all four folders into a fifth outer folder named with your Andrew id.**
- **You will put the folder named with your Andrew ID into your handin folder on the server. Use the Exam1 folder.**
- **You will zip the folder named with your Andrew ID and mail it as an attachment using the subject line:**
  Exam 1
  to Jim at:
  jr2u@andrew.cmu.edu
- **You should mail a copy to yourself as backup security.**

There is partial credit for all four question – do not erase non-working code.

Question 1 5 points Locating Positions with Expressions

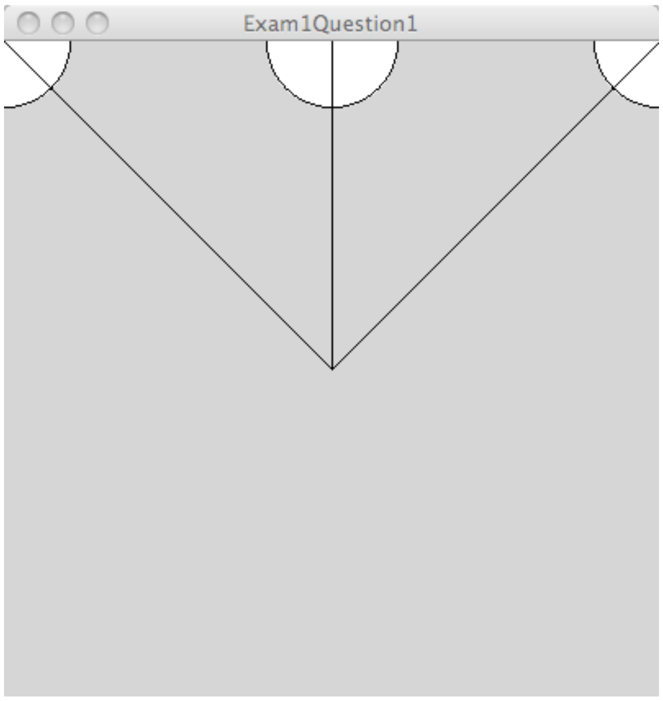
Write a program without **setup()** and **draw()** functions (as you wrote the first two homeworks) and with no user input to draw the figure shown below.

The code must use variables and expressions to locate the figures. It must not use magic numbers.

The figure must adjust to fit into different window sizes without any recoding other than changing the window size.

The color of the **background**, the **fill**, and the **stroke** are the default colors – do not worry about them. The **strokeWeight** has a default of 1. Do not alter these values.

Your figure should look like this:



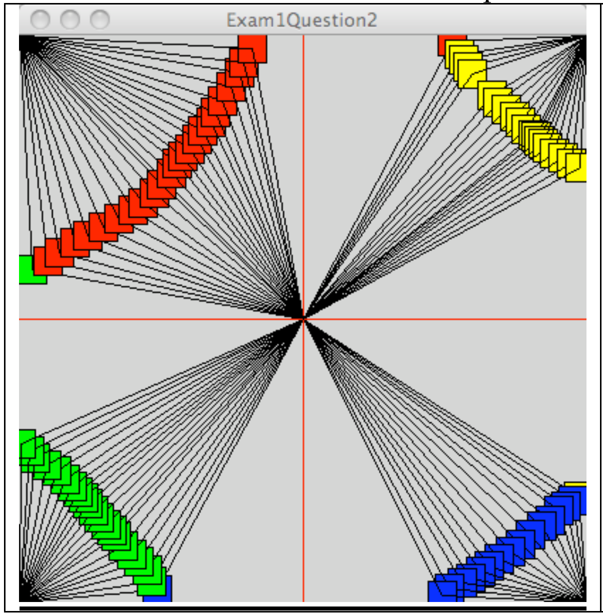
Question 2 10 points Choosing What to Do

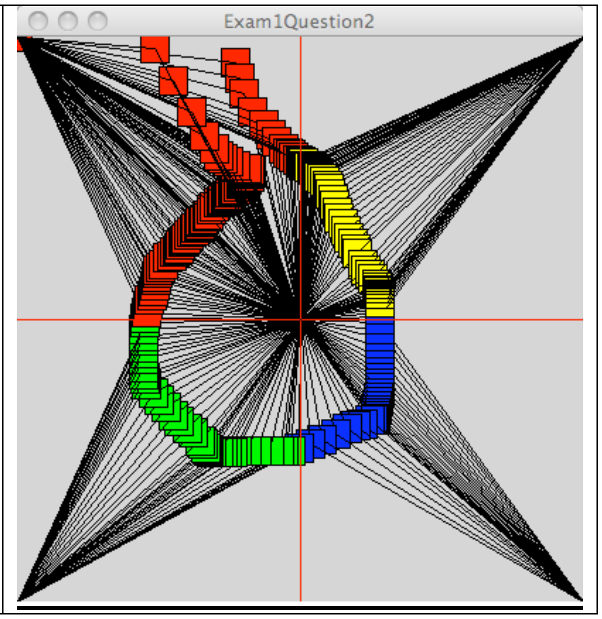
Write a program using **`setup()` and `draw()`** functions.

The program must do the following:

- Draw a small sized filled rectangle in every frame that is at the location of the mouse. The rectMode() can be the default CORNER mode or you can change it to CENTER. The code used to make the figures had the rectMode( ) set to CENTER. It does not matter.
- The color of the rectangle must be based on the quadrant in which the rectangle is drawn. The four quadrants are the upper left, upper right, lower left, and lower right. The colors are up to you. We suggest red (255, 0, 0), green (0, 255, 255), blue (0, 0, 255), and yellow(255, 255, 0).
- Draw two lines must to some point within the rect.
  - one line must extend from the center of the window
  - one line must extend from the corner closest to the rectangle
- Do not draw the red lines – they are there to show you the quadrant boundaries.

Here are two different runs of a sample solution:





Notes:

- There will be a rectangle in the upper left corner until you move the mouse into the frame.

- The first rectangle drawn in a new quadrant may be the wrong color – this is fine.

Question 3 15 points User Input

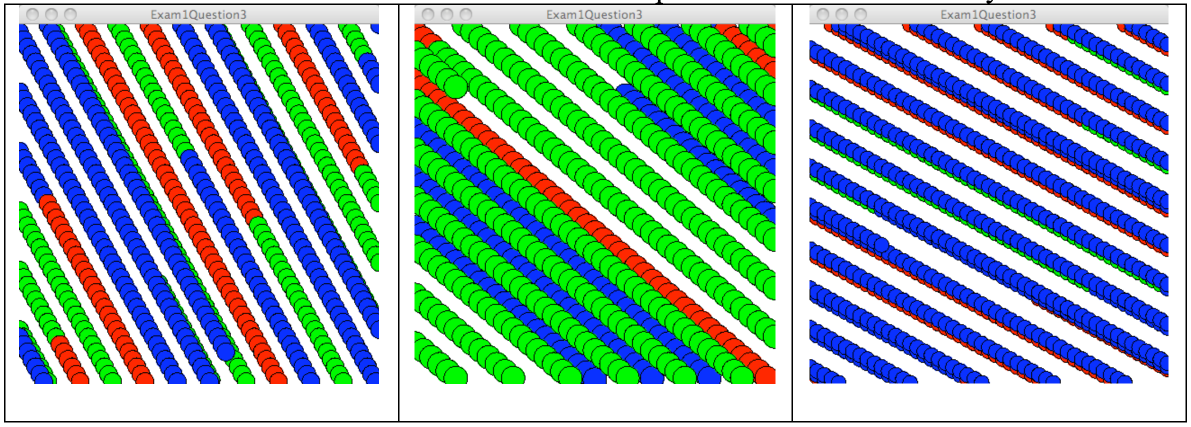
Write a program using **`setup()`** and **`draw()`** functions.

The program must do the following:

- Declare variables to store the position, size, and deltaX and deltaY values for a circle or square. There must be two delta variables.
- Initialize the variables to random values as follows:
  - x and y to any position in the frame
  - size variable/s between 5 and 30
  - delta variables between 3 and 10

  Use the **`random( )`** function to do this.
- Initialize the **`fill`** to white (255).
- Each frame must draw the circle or square and then move it to a new position that is computed by adding the delta values to the x and y variables.
- If the circle or square moves off the right side or the bottom side, it must be wrapped back to the left side or the top side.
- If the user presses a key, change the fill as follows:
  - the '**`r`**' key changes the fill to red (255, 0, 0)
  - the '**g**' key changes the fill to green (0, 255, 0)
  - the '**b**' key changes the fill to blue (0, 0, 255)
  - any other key press, change the fill to white (255).
- If the user presses the mouse, the variables used to draw and move the square or circle are assigned new random values as listed above.
  DO NOT CALL **`setup()`** TO DO THIS!

Here are three different screens after the user has pressed the mouse or a key:



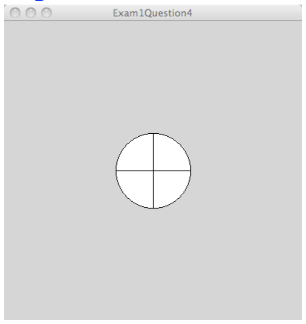
Question 4 - 20 points Defining a Function with Arguments

Write a program using **`setup()`** and **`draw()`** functions.

You **MAY NOT USE** GLOBAL VARIABLES*.

The program must do the following:

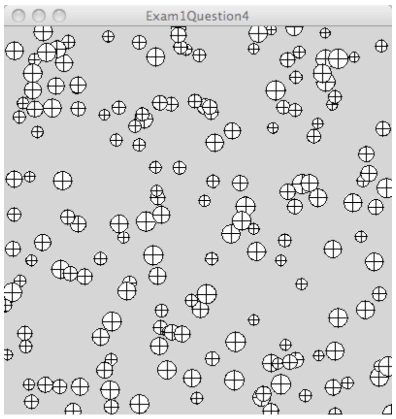
- define a function named **`figure()`** that draws the figure shown below



- The function must have three arguments:
  - two for the **`(x, y)`** coordinates
  - one for the size of the figure.
- The figure is composed of a circle and two lines.
- The anchor point location of the **`(x, y)`** coordinates is up to you.
- The **`stroke`**, **`strokeWeight`** and **`fill`** are the default values.
- One call of the **`figure( )`** function results in ONLY ONE figure being drawn.
- The function **`draw()`** calls the **`figure()`** function using random numbers to position the figure and a reasonable random number for the size of the figure. Here is what your **`draw()`** function must look like:

```
void draw ( )
{
  figure(random(0, width), random(0, height), random(10, 20));
}
```

Here is a run of a sample solution:



* If you cannot write the function **`figure( )`** with arguments as required, you may write it using global variables. This will be a point deduction but it will be **MUCH** better than taking a zero for the question.