Computer Science 3-4
Problem Set #13: Drawing Shapes with SFML

We will be using the Simple Fast Media Library (known as SFML) for the rest of the year to handle all of our graphical needs. It has some really powerful features, including using OpenGL for graphics, handling keyboard and mouse events, playing audio through the sound card and even some networking features. Throughout these lessons, you may find it useful to visit www.SFML-dev.org . This site has all of the documentation for the SFML language and has a wide variety of tutorials if you wish to work on your own. In particular, there are excellent tutorials on installing SFML on your own.

Note: Setting up your first SFML project can be a bit tricky, check out the video on the site for a walkthrough: http://www.youtube.com/watch?v=vt0CiMGzBo8&safe=active .

Note: The examples in the section will put a prefix of "sf::" on most of the commands we use. If you would like to avoid doing this, just add "using namespace sf" at the top of your program.

The Basic Window

Before we draw any shapes, we need to make a *rendering window*. This is simply telling the program that we are going to be drawing in a specific place. We will be using this window as the skeleton of almost everything we do while we play around with graphics, so get used to it!

```
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(800, 600), "SFML Saves!");


    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();

        window.display();
    }

    return 0;
}
```

For most drawing activities, we will be inserting code in between the window.clear() and window.display(). Each of these commands needs to appear only once in your code…calling window.display() too frequently can crash your computer!

We will be using Circles, Rectangles and Lines to begin the lesson; here is the general format for each of these objects:

1. Create the Shape using the appropriate settings (see below for specific examples)
2. Position/Modify the shape properties (color, location and such)
3. Draw to the screen

Here is a code snippet that draws a few shapes to the screen:

```
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(800, 600), "SFML Saves!");
    //Draw a Circle with radius 50
    sf::CircleShape myCircle(50);
    // Place circle in position
    myCircle.setPosition(100,100);
    // Make it red!
    myCircle.setFillColor(sf::Color(255,0,0));
    //Draw a Rectangle define upper left corner
    sf::RectangleShape myRect(sf::Vector2f(120, 50));
    //Give it a length and width
    myRect.setSize(sf::Vector2f(100, 50));
    myRect.setFillColor(sf::Color(0,255,0));
    myRect.setPosition(400,300);

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        window.clear();
        window.draw(myCircle);
        window.draw(myRect);
        window.display();
    }
    return 0;
}
```

Notice that we are creating a sub-object of "sf::Shape" with the name CircleShape and RectangleShape and then calling specific functions to set the object's properties. If you want to draw a line, then you can use two methods:

Method 1: Draw a very skinny rectangle…the easy way but is clumsy in that it can be very hard to position.

Method2: Use a *vertex array*…this is fancy (and you may want to learn more about it by checking out this tutorial: http://www.sfml-dev.org/tutorials/2.1/graphics-vertex-array.php). I will talk about this more in class and in the next packet!

Regular Polygons – The CircleShape object has something we call an *overloaded constructor*, this is a fancy way of saying that the function will behave differently if you send it a different set of arguments. For example:

-------insert example

The window.draw() function doesn't actually put anything to the screen (that's what window.display() does), instead it builds up a list of what needs to get drawn when we update the screen.

One important thing to remember is that the coordinate axes are centered around the upper left corner of the window and get larger as we go down to the right (so the top left corner is always 0,0 and the bottom left will be 800,600 in our case). If you need help on how to make colors with the R, G, B format, check out this link: http://www.colorschemer.com/online.html.

Now for a few problems!

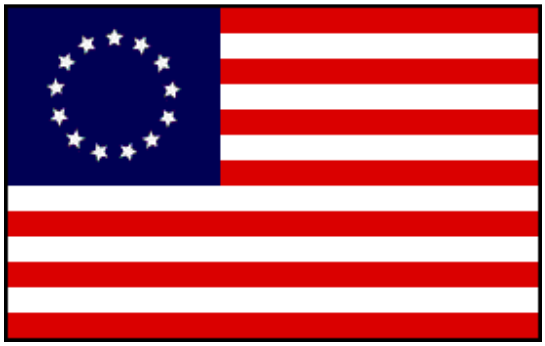
1. Create a program that draws a window and then draw four of each of the basic shapes to the screen (i.e. 4 different circles, 4 different lines, etc.). Use different positions, colors, outlines, etc.
2. Write a program that asks the user how many circles they would like to see and draws that many circles of random size, color and position to the screen.
3. Write a function that will take three points from the user and draw a triangle to the screen. Use this to write a program that does the following:
    a. Takes three points from the user and determines if the triangle is a right triangle.
    b. Finds the area of *any* triangle
    c. Draws the triangle to the screen *and* a rectangle of equal area
    d. **Challenge** – if the triangle is a right triangle, draw the a^2 , b^2 and c^2 squares on the respective sides of the triangle.

4. **A simple animation!** – We can animate objects by displaying them, clearing the screen, calculating a new position and redrawing them. Use this idea to animate a circle that moves across the screen (note if you don't clear the screen between draws it will make a "snake" trail). Hint: Write a function that gives you the x and y position as a function of time (or steps through a loop) to avoid writing a ton of code.

We can build custom shapes by using the ConvexShape class. Play around with setting the points in different orders…you can get some interesting shapes this way!

```
29  // create an empty shape
30  sf::ConvexShape convex;
31  // resize it to 5 points
32  convex.setPointCount(5);
33
34  // define the points
35  convex.setPoint(0, sf::Vector2f(0, 0));
36  convex.setPoint(1, sf::Vector2f(150, 10));
37  convex.setPoint(2, sf::Vector2f(120, 90));
38  convex.setPoint(3, sf::Vector2f(30, 100));
39  convex.setPoint(4, sf::Vector2f(0, 50));
40  convex.setFillColor(sf::Color::Magenta);
41  convex.setPosition(500,300);
```

5. Using the ConvexShape class, write your own function that will generate a five-pointed star centered on a point the user requests and of a specified width in pixels (note: you will need to do a bit of geometry to figure out where all your points will be!).
6. Use your function from #5 to write a function to draw a "circle of stars" like we see here in the 13 colonies flag.



The user should be able to request how many stars should be in their circle and where the center of the ring is located. Your function should scale the stars to an appropriate size for the number requested.

7. **Challenge x2: Go America! –** Use what you've learned to make an animated version of the 13 colonies flag complete with flashing colors and a *spinning* circle of stars (if you want to get really fancy, figure out a way that you can get each individual star spinning as well as the whole ring).