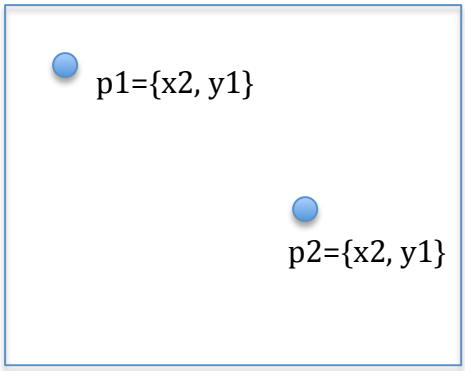
Tutorial challenges

Today will do some collision detection – how to compute when two object bump in to each other!

Basic math: the distance between two 2-D objects p1 and p2 is:



$$distance = sqrt(x_2 - x_1)^2 + (y_2 - y_1)^2)$$

OVERVIEW – We will start with a template almost exactly from last week's tutorial with the moving circle.

Then we will make a "shooter" and detect when the bullets het the circle!

1. Start with template (essentially from last tutorial)
    a. All of your new code today will go between the section markers:
    //==================================

2. Create the graphics objects we need:
    a. Notice I have provided a distance() function that takes two arguments that are assumed to have an 'x' and a 'y' attribute (I think of such objects as "points") . Examine it and see how it implements the distance formula above

    b. Create a variable "shooterPt" as an object for the location of your shooter, with its x and y attributes set to position the shooter at the bottom center of your paper.

    c. Create a graphical shooter by drawing a FAT line (paper.path) from the shooterPt with a length of 30. The path is just a line between two points, and you can make it fat by setting a big (e.g. 50) stroke-width attr. Don't forget to give it a fill, too.

    d. Now create a bullet (a circle) or radius 25 positioned on top of your shooter. Then add your own attributes (not Raphael attributes) to the

bullet for xrate, and yrate (both initialized to 0), and posx and posy (initialized to the same values as your shooterPt attributes).

4. Our next step will be to make the shooter shoot when we click the shooter. This is a GREAT place to STOP coding and THINK and TALK about how you might make this work. See if you can work out a design for it – what are the few different components you will need, and where and how might you write the code?

. . . . . .

3. First, lets add code to our draw() function that will make the bullet move when it is shooting.
    a. update bullet.posx by adding bullet.xrate
    b. update bullet.posy by adding bullet.yrate
    c. update the graphical bullet object attributes with your newly computed position variable.
    d. Run your code. The bullet won't move because we set its 'rate' variables are still 0, but the code should run without crashing!

4. Now let make the shooter shoot:
    a. Add a listener for clicks to your shooter (which should be visible behind your bullet) that we will use to make the shooter shoot. For now, just print a console message and check that you are getting the clicks.

    b. Now write a little "shoot()" function. To make it shoot, lets reset its posx and posy attributes to the original shooterPt location, and then set its yrate (-5 works pretty well – negative so that it shoots up!)

    c. Call the shoot function from within your shooter click listener.
        i. Check your work!!!!!!

5. Now your are shooting, but how do you know when your shot was successful?
    a. Explain it in English first to your roomies
    b. How would you write it in code?
    c. Our strategy will be to test to see if the two objects are overlapping. To do that we will write a function that returns true if they are, false otherwise.
    d. Write a function called "circleBump" that takes to arguments, both assumed to be Raphael circles.
        i. compute the distance 'd' between their centers by calling the distance() function with two object literals with their x and y attribute values taken from the circle centers.
        ii. if 'd' is less than the sum of the circle radii, well the, they must be overlapping! So return true, otherwise return false.
            1. Check your work with console messages.
    e. In your draw function, call your circleBump function with your bullet and the target circle. If they bump (if circleBump returns true), change the color of the circle to one color, if they are not bumping, set it to its original.

Cool! You wrote a shooter game with collision detection!!!