Lab 04 Bonus Worksheet 1: A Fading Circle

In these notes you will learn:

- How to make an object fade away by changing its transparency.
- How to use the constrain function.
- How to write your own version of the constrain function.

Introduction

So far, all the animations we've created move an object by changing its position. In this note we will see another kind of animation: fading. When the user clicks on the screen a big blue circle appears. It then begins to fade away until it is invisible.

Organising Our Demo Program

The first step in writing any program is to decide exactly what it should do. Since we are writing a demonstration of a new kind of animation, we want to keep it relatively simple. So let's agree that the program works as follows: when the user clicks on the screen, a circle appears at the click-point and then fades away to invisibility.

We'll need variables to store its (x, y) centre. The circle won't be moving or changing shape, so we don't need `dx`, `dy`, or `r`.

How do we get the circle to fade? Fading means that the circle starts out with some totally opaque (i.e. not see-through) colour, and then slowly becomes more transparent, finally disappearing completely.

Recall that Processing lets you specify a colours alpha value, which sets the opacity of a colour. Alpha values in Processing range from 0 to 255, where 0 is totally transparent (i.e. invisible), and 255 is totally opaque (i.e. not transparent at all).

So we will implement fading by slowly decreasing the alpha value of the circle's fill colour, resulting in slowly increasing the transparency.

First Draft: Just the Circle

Let's write a program that displays a big blue circle wherever the user clicks with the mouse. There is no fading yet; we'll add that next after we get this working:

```
float x; // X position of the circle
float y; // Y position of the circle

void setup() {
  size(500, 500);
  smooth();
}

void draw() {
  background(255);
  noStroke();
  fill(0, 0, 255);
  ellipse(x, y, 150, 150);
}

void mousePressed() {
  x = mouseX;
  y = mouseY;
}
```

Recall that the `mousePressed()` function is automatically called by Processing whenever the user clicks a mouse button. When it's called here, it sets x and y to be the position of the mouse, and so when `draw()` is called the circle will be drawn centred at the most recent mouse-click.

Adding Fading

Now let's add the fading effect. We'll need a variable to keep track of the circle's opacity. Let's agree that opacity ranges from 0 (totally transparent) to 1 (totally opaque). A range of 0 to 1 is pretty common because it makes it easy to talk about percentages, e.g. 0.73 is exactly 73% of the way between 0 and 1.

We will also need a variable that tells us how much to decrease the opacity on each call to `draw()`. In other words, we need a variable that stores how fast the opacity change. We call this variable `dOpacity` following the convention of starting "speed" variables (like dx and dy) with the letter d (for "delta").

Let's declare those two variables:

```
float opacity;  // Circle opacity, ranges from 0 to 1 (0: totally transparent)
float dOpacity; // Change of the opacity per frame
```

Now we can add the fading effect. We need to make only a few changes:

Whenever the user clicks the mouse, the circle is drawn at a new location totally opaque. Thus we need to set `opacity` to 1 at the end of `mouseClick()`:

```
void mousePressed() {
  x = mouseX;
  y = mouseY;
  opacity = 1; // make the circle visible
}
```

Whenever the circle is drawn, its fill colour must be drawn taking the opacity into account. This is not completely straightforward. For instance, this doesn't work:

```
void draw() {
  background(255);
  noStroke();
  fill(0, 0, 255, opacity);  // Wrong!
  ellipse(x, y, 150, 150);
}
```

The problem is that the fill function expects a value in the range 0 to 255 (you learn this by reading the `fill()` function documentation). But opacity ranges from 0 to 1.

We could, of course, change our previous decision and let opacity range from 0 to 255. That would be fine for this one program. But in general, it's probably not a good idea. In large programs, you may have many different ranges, and if these ranges all have different begin and end points you will soon find it takes a lot off mental effort to keep track of them. It's simpler if you require that all ranges start at 0 and end at 1.

So let's keep the range of transparency as it is, and instead use the map function to convert opacity into a corresponding value in the range 0 to 255:

```
void draw() {
  background(255);
  noStroke();

  // convert opacity from 0..1 to alpha value from 0..255
  float a = map(opacity, 0, 1, 0, 255);
  // draw circle
  fill(0, 0, 255, a);
  ellipse(x, y, 150, 150);
}
```

This works! The expression `map(opacity, 0, 1, 0, 255)` returns the `a` number in the range 0 to 255 that corresponds to the same value of opacity from 0 to 1.

After the circle is drawn, we need to change opacity by adding dOpacity to it:

```
void draw() {

   ...

  // change opacity
  opacity += dOpacity;
}
```

Since we've decided to add `dOpacity` to `opacity` (instead of subtracting it), we must remember to initialize `dOpacity` in `setup()` to a negative value in order to get a fading effect.

An important detail arises here. We've decided that opacity ranges from 0 to 1, but the statement `opacity += dOpacity` does not respect that range. For instance, if `dOpacity` is negative, then if you add `dOpacity` enough times, `opacity` will become negative, i.e. it will be outside of the agreed-upon range of 0 to 1.

Sometimes this might not matter. In fact, in this program, it doesn't matter because it turns out that if you call fill with a negative alpha value then the alpha is treated as if it were 0.

But in general it is unwise to rely on functions handling errors the way you want. So we should make sure that the value of `opacity` is always in the correct range. The easiest way to do that in Processing is to use the constrain function:

```
void draw() {
  background(255);
  noStroke();

  // convert opacity from 0..1 to alpha value from 0..255
  float a = map(opacity, 0, 1, 0, 255);
  // draw circle
  fill(0, 0, 255, a);
  ellipse(x, y, 150, 150);

  // change opacity, and keep between 0 and 1
  opacity += dOpacity;
  opacity  = constrain(opacity, 0, 1);
}
```

The expression `constrain(opacity, 0, 1)` always returns a value from 0 to 1. If opacity happens to be between 0 and 1, then its value is returned. Otherwise, if opacity is less than 0, 0 is returned; if opacity is greater than 1, 1 is returned. In fact, it is not hard to write your own version of `constrain`:

```
float myConstrain(float x, float lo, float hi) {
  if (x < lo) {
    return lo;
  } else if (x > hi) {
    return hi;
  } else {
    return x;
  }
}
```

Programming Questions

1. Add a background image to the fading circle program. As the circle fades away, the underlying image should become clearer.
2. Modify the fading circle program so the circle fades in (instead of fades out). When the user clicks with the mouse, the circle begins invisible and then slowly becomes fully opaque.
3. Modify the fading circle program so that after the transparency of the circle fades to 0, it starts to fade back in, and then it fades back out, and so on forever. The circle should appear to slowly pulse.
4. Read about the `tint()` function and then modify the fading circle program to make an arbitrary image (loaded from a file) fade away.

The Fading Circle Program

```
float x;        // X position of the circle
float y;        // Y position of the circle
float opacity;  // Circle opacity, ranges from 0 to 1 (0: totally transparent)
float dOpacity; // Change of the opacity per frame

void setup() {
  size(500, 500);
  smooth();
  // initialise variables
  opacity  = 0;
  dOpacity = -0.1;
}

void draw() {
  background(255);
  noStroke();

  // convert opacity from 0..1 to alpha value from 0..255
  float a = map(opacity, 0, 1, 0, 255);
  // draw circle
  fill(0, 0, 255, a);
  ellipse(x, y, 150, 150);

  // change opacity, and keep between 0 and 1
  opacity += dOpacity;
  opacity  = constrain(opacity, 0, 1);
}

void mousePressed() {
  x = mouseX;
  y = mouseY;
  opacity = 1; // make the circle visible
}
```