Lab 11

- Fill in steps 2a and 2b so that each boid chooses between four behaviors: *avoid obstacles*, *avoid collisions* with other boids, *seek*, and *wander*. The particular action is chosen as follows:
  - If there's an obstacle in range, the boid should avoid it.
  - Otherwise if there's potential for colliding with another boid, avoid the collision.
  - Otherwise if the boid is near the mouse (within 200 pixels), seek on the mouse position.
  - Otherwise wander.

use dist for the "within … pixels" or "more than … pixels" conditions

for the conditions outlined in red, use the magnitude of the steering vector

- Fill in prey steps 2a and 2b (in the "update prey" section) in `draw()` so the prey behaves as follows:
  - If the prey boid is within 250 pixels of the predator, it should evade the predator and flock with the other prey boids. (This means four behaviors are active in this step — *evade*, *separation*, *alignment*, and *cohesion*.)
  - Otherwise if the prey is not near the food (more than 100 pixels away) and there's potential for colliding with another prey boid, avoid the collision.
  - Otherwise if the prey boid is hungry (`hunger` is 0 or lower) and is within 300 pixels of the food, it should arrive at the food.
  - Otherwise the prey boid should wander.

For collision avoidance and obstacle avoidance, the computed steering vector has a magnitude of 0 if there is nothing to pursue or avoid. You can get the magnitude of a `PVector v` with `v.mag()` — so, for example, if you compute the obstacle avoidance steering vector and it has a magnitude of 0, there's no obstacle to avoid.

```
// compute the evade steering vector
//  pos, vel - position and velocity of boid
//  maxspeed - boid's max speed
//  pursuerpos, pursuervel - position and velocity of pursuer
PVector computeEvade ( PVector pos, PVector vel, float maxspeed, PVector pursuerpos, PVector pursuervel ) { ..

// compute the pursue steering vector - pursues the nearest quarry within the boid's neighborhood
//  pos, vel - position and velocity of boid
//  radius, angle - define this boid's neighborhood
//  maxspeed - boid's max speed
//  quarrypos, quarryvel - positions and velocities of potential quarries
PVector computePursue ( PVector pos, PVector vel, float radius, float angle, float maxspeed, PVector[] quarryp
```

These functions return a steering vector with a magnitude of 0 if there is nothing in sight.

Lab 12

- loading images – 2 steps:
  - add the image file to the sketch: Sketch → Add File…
    - this copies the image file into the sketch folder so that everything the sketch needs to run is contained together
  - use only the filename with `loadImage()` e.g. `loadImage("marmot.jpg")`
    - do not use an absolute pathname (such as `/classes/cs120/images/marmot.jpg`) because the sketch won't run if copied to a different computer system
    - do not use a URL (such as `https://math.hws.edu/bridgeman/courses/120/f24/inclass/images/marmot.jpg`) because the sketch won't run if that URL isn't available

Lab 12

- use the patterns from class

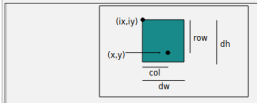
- Modify the drawing function to instead take the shape colors from the image. Put the `loadPixels()` step at the beginning of your drawing function instead of in `setup()`. Refer to the slides and examples from class for the pattern and formulas for converting a point (x,y) in the drawing window to a pixel (row,col) in the image and then getting the color of that pixel.

Wed Reading:

- *Learning Processing*, sections 15.5-15.6
- *Learning Processing*, section 15.10

Materials from class:

- slides: using images as a source of colors
- examples: (using images as a source of colors)
  - colored dot
- handout: in-class exercises (solutions: #1, #2)
  - images to use
  - `starter dot` (for #1)



important: to avoid confusion, use (x,y) to refer to coordinates on the screen and (row,col) to refer to coordinates within an image

- assumptions/notation
  - image has size `img.width` × `img.height`
    - replace `img` with the name of the image variable
  - image is positioned in the drawing window with corner (`ix`,`iy`) and dimensions `dw` × `dh`
- to transform a drawing window coordinate (`x`,`y`) into an image pixel coordinate (`row`,`col`) –
  - `row = (y-iy)*img.height/dh`
  - `col = (x-ix)*img.width/dw`
- pixels are stored in the array `img.pixels`
  - pixel at position (`row`,`col`) is stored in slot `loc = row*img.width+col`

```
// draw colored spot following the mouse - color is taken from the ima
//  under the mouse
{
  // (row,col) corresponding to the mouse's location in the image
  //   row = (y-iy)*img.height/dh, col = (x-ix)*img.width/dw
  //  (x,y) is the mouse position
  //  image's corner (ix,iy) is at (0,0),
  //  image's displayed size dw x dh is width x height
  int row = (mouseY-0)*img.height/height;
  int col = (mouseX-0)*img.width/width;
  // location in pixels array corresponding to (row,col)
  int loc = row*img.width+col;

  fill(img.pixels[loc]);
  stroke(0);
  ellipse(mouseX,mouseY,40,40);
}
```

Lab 12

- use the patterns from class

- Create a filter function for your image filter — name it *`filter`* (substituting the name of the filter for *filter*) and include the parameters listed in the description below. The body of the function will be very similar to the brighten and green/blue swap filters from class — just the details of how you compute the destination colors will be different.

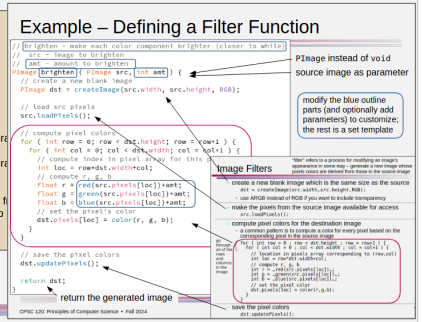
Mon Reading:

- *Learning Processing*, sections 15.7-15.8
- *Learning Processing*, section 15.9 (optional)

Materials from class:

- slides: comments on lab 10
- slides: generating images and image filters
- examples:
  - random image #1 (generating images, genera once)
  - random image #2 (generating images, genera every frame)
  - brighten #1 (image filter, apply filter once)
  - brighten #2 (image filter, apply filter in every f
- handout: in-class exercises (solutions: #1 — swap filter, #2 — generating gradient image)
  - images to use

Example – Defining a Filter Function



Lab 12

- follow the instructions in the lab handout
  - include all the specified parameters for the drawing functions for the artistic effects

effect	example	function parameters	how it works
pixellate		• source image • position and size of the drawing area • rectangle size	Cover the drawing area with rectangles, taking the fill and stroke colors of each rectangle from the image pixel corresponding to the center of the rectangle. You may need to turn off smoothing to avoid slight gaps between the rectangles. Note: be careful not to draw rectangles that extend past the edge of the image. Your function should work with any size rectangle, but experiment with different values to find one with a pleasing effect to use in your sketch.

  - leave a border around the edge – the image should not fill the window

Your task is to create a sketch named **lab12a** which applies an artistic effect to an image. Choose one of the artistic effects described in the "Artistic Effects" section below.

Leave a border of the background showing as in the example. (The background can be any color; it doesn't have to be black.) Create a drawing function for the artistic effect.

To create this sketch:

Lab 12

- credit the source of the images used in #3

You are also free to use other images but **you must have permission for all images that you use**. This means that the image is one of the provided images above, a picture you took or created, a picture someone else took or created and gave you permission to use, or an image that you found on the Internet that explicitly allows you to use it. There are many public domain images that are fine to use, but not every image is public domain! Include a comment in your sketch identifying the source for each image that you use. (Give the URL if the image came from the Internet.)

Midterm Project

- the main issue was missing requirements
  - read the handout carefully as you are planning
  - check over the handout again along the way and when you are done

Midterm Project

- commonly missed elements

- Include a comment at the beginning of the sketch with your name and a description of your sketch — identify the sequence of actions, what triggers each new stage, and the final goal. Also identify how you satisfy the requirements marked with (*) above — include a comment at the beginning of the sketch or just before the relevant section of code, as appropriate.
  - Put only drawing into drawing functions — do updates in `draw()`, not in a drawing function.
- At least three actions that last for two or more stages (but not every stage). In the demo, the these actions are:
  1. The pendulum keeps swinging after it hits the floating platform. (stages 3-5)
  2. The floating platform swings after it is hit by the pendulum. (stages 4-5)
  3. The red ball keeps bouncing after it falls off the end of the ramp. (stages 3-5)

  It is not necessary for a multi-stage action, once started, to last for all of the remaining stages (though that is the case in the demo). For example, it would be fine for an action to only last for stages 2 and 3.
- At least two instances of some compound thing (three or more shapes) drawn by a function with appropriate parameters. The instances must be interactive or animated in different ways; not interactive or animated is a possibility for one instance. (Not satisfied by the demo.)