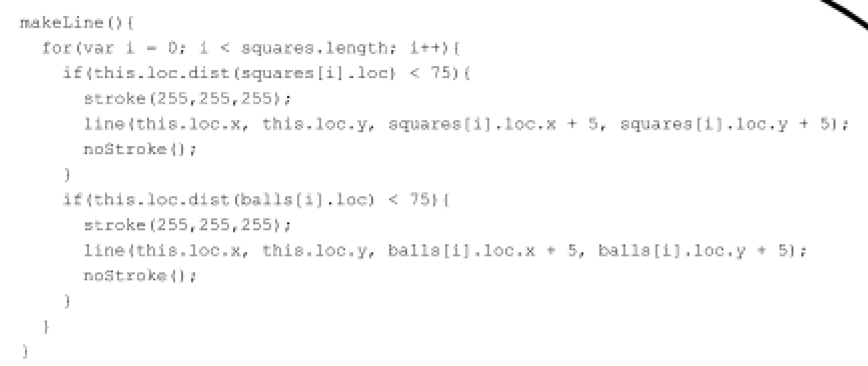
Austin Matel
Mr. Ettlin
APCS Principles, Period 1
5 March 2020

Create Task Answers

2a.) This program was created in p5js and is made to entertain the user. This program consists of many small shapes on the screen that are attracted and repelled from the mouse. When moving the mouse, the triangles, circles, and squares should turn away from the mouse and be repelled when they are a certain distance from the mouse. Once the objects are out of the repelling radius, the shapes should turn back towards the mouse and move towards it. They will continue to move towards it until it reaches the repelling radius. If the objects get close to each other, a line will be created to connect the two shapes. If the mouse is moved towards the shapes with lines and the shapes spread out far enough, the line will disappear until the shapes come close together again. To reset, simply refresh the page and continue to move the mouse over the canvas.

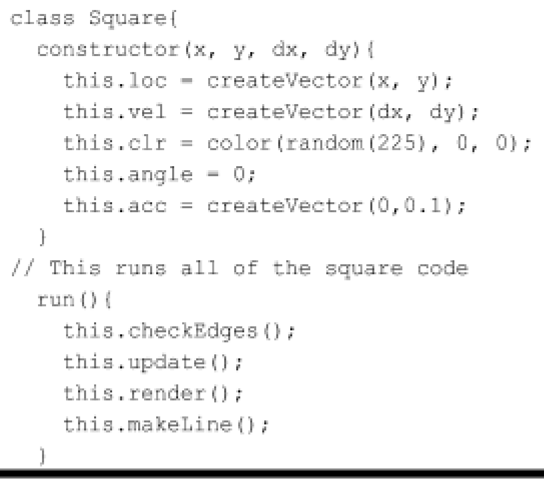
2b.) For planning, I planned to make many shapes which would be attracted and repelled by a central point. I first created the different shape classes as well as a boid class to represent the mouse. Then I gave each object a speed so that they could move (discluding the boid, it was at the mouse). Then I created the properties of the repelling and attracting. To do this, I copied some code that I had created previously, adding both the attractor and repeller code to the boid class. Then I made it so lines were created in between shapes when they came close. One problem I had was that the squares did not move when I called their run function. I checked the inputs being fed into the class and found that I had forgotten to feed in a velocity. Another problem I faced was that the lines did not show up when I first added the makeLine algorithm. I asked what my friend thought was the issue and they looked through that portion of my code. It turns out that I didn't have a stroke color. I changed the stroke color to white and the lines appeared.

2c.) This algorithm creates a line between the shapes of the program when they are close to each other. The two if statements inside are the algorithms within the algorithm and they use booleans and math to determine if the shapes are close enough to connect them with a line. The first if statement connects squares to the triangle and the second if statement connects the triangles to the balls. Without this code, the art would be only shapes surrounding a spot and would get rid of a major user interaction aspect.



```
makeLine(){
  for(var i = 0; i < squares.length; i++){
    if(this.loc.dist(squares[i].loc) < 75){
      stroke(255,255,255);
      line(this.loc.x, this.loc.y, squares[i].loc.x + 5, squares[i].loc.y + 5);
      noStroke();
    }
    if(this.loc.dist(balls[i].loc) < 75){
      stroke(255,255,255);
      line(this.loc.x, this.loc.y, balls[i].loc.x + 5, balls[i].loc.y + 5);
      noStroke();
    }
  }
}
```

2d.) This abstraction is a class that holds all of the code for the square shapes on the screen. This code makes the squares appear on the screen, makes them move, and makes them attracted or repelled by the mouse. The mathematical and logical concepts lie within the methods which are being called in the run function ("this.checkEdges" and "this.update" for example). This abstraction contains all of the necessary code for the shape and makes my code less complex so that I don't have to repeat this code for all of the shapes on the screen and then run it all.



```
class Square{
  constructor(x, y, dx, dy){
    this.loc = createVector(x, y);
    this.vel = createVector(dx, dy);
    this.clr = color(random(225), 0, 0);
    this.angle = 0;
    this.acc = createVector(0,0.1);
  }
// This runs all of the square code
  run(){
    this.checkEdges();
    this.update();
    this.render();
    this.makeLine();
  }
```