CmpE 49G - Project 2

Generative Art with Perlin Noise

GitHub Repository: github.com/egirgin/cmpe49g

Name & Surname: Emre Girgin

ID: 2016400099

What is Generative Art ?

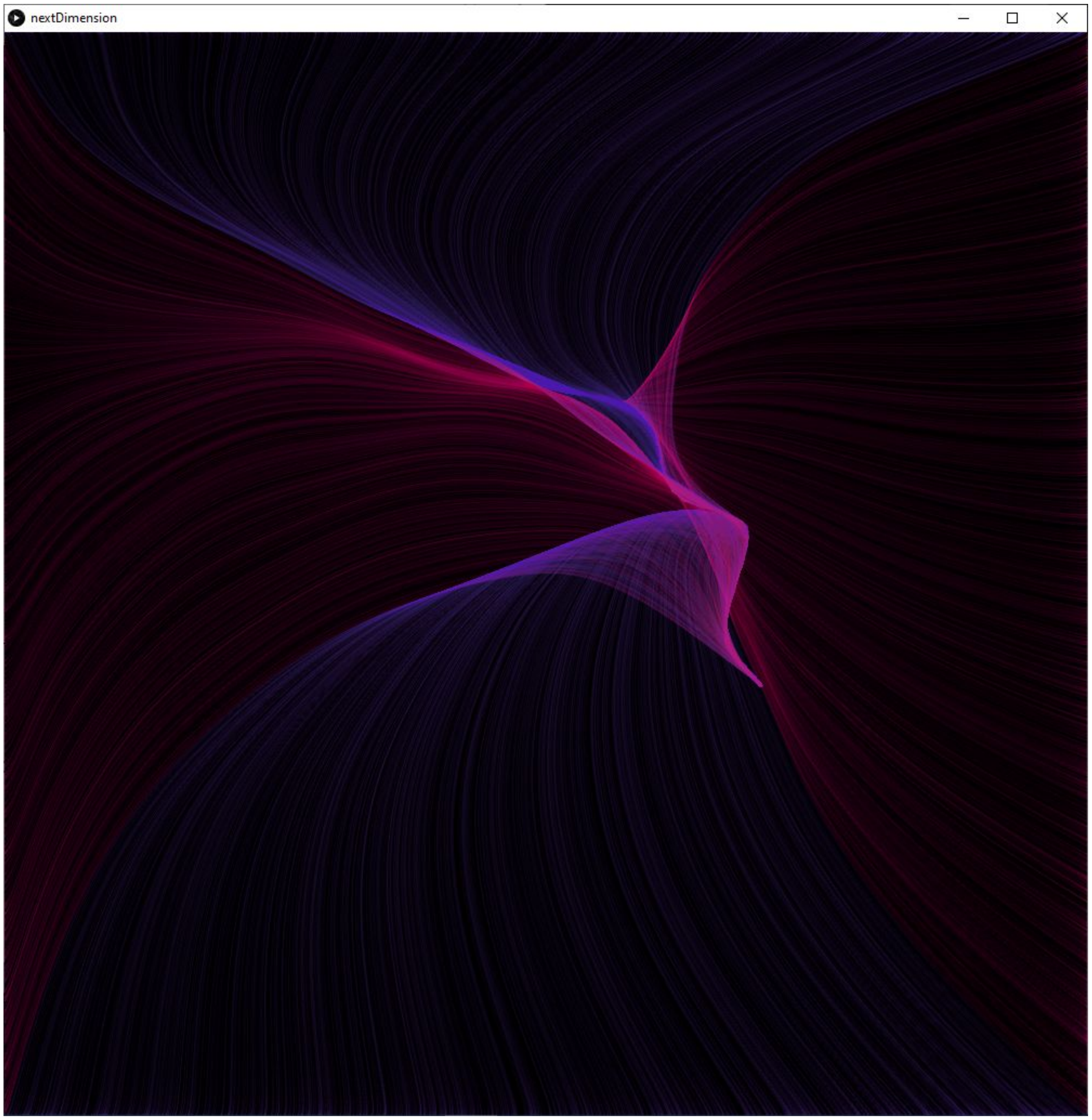
Generative art is a piece of art that has been "generated" by an autonomous system. In contrast to traditional art, the artist does not need to be a human or a creature. (see the art painted by animals.) Today, thanks to the processing power of novel computers, we are able to create more detailed and fascinating visual products that can be considered as art or generative art. However, the history goes all the way back, up to the 1960s, to the invention of computer graphics, according to Wikipedia. Computers can process huge amounts of particles and their interactions and demonstrate a wide range of color palettes. Hence, by combining those, one can create an original piece of art. In addition, thanks to some of the deep learning techniques like Neural Style Transfer, we are able to transform a photo we took, into a painted version of it with the style of a famous painting. I think this should also be considered as generative art.

What is Perlin Noise ?

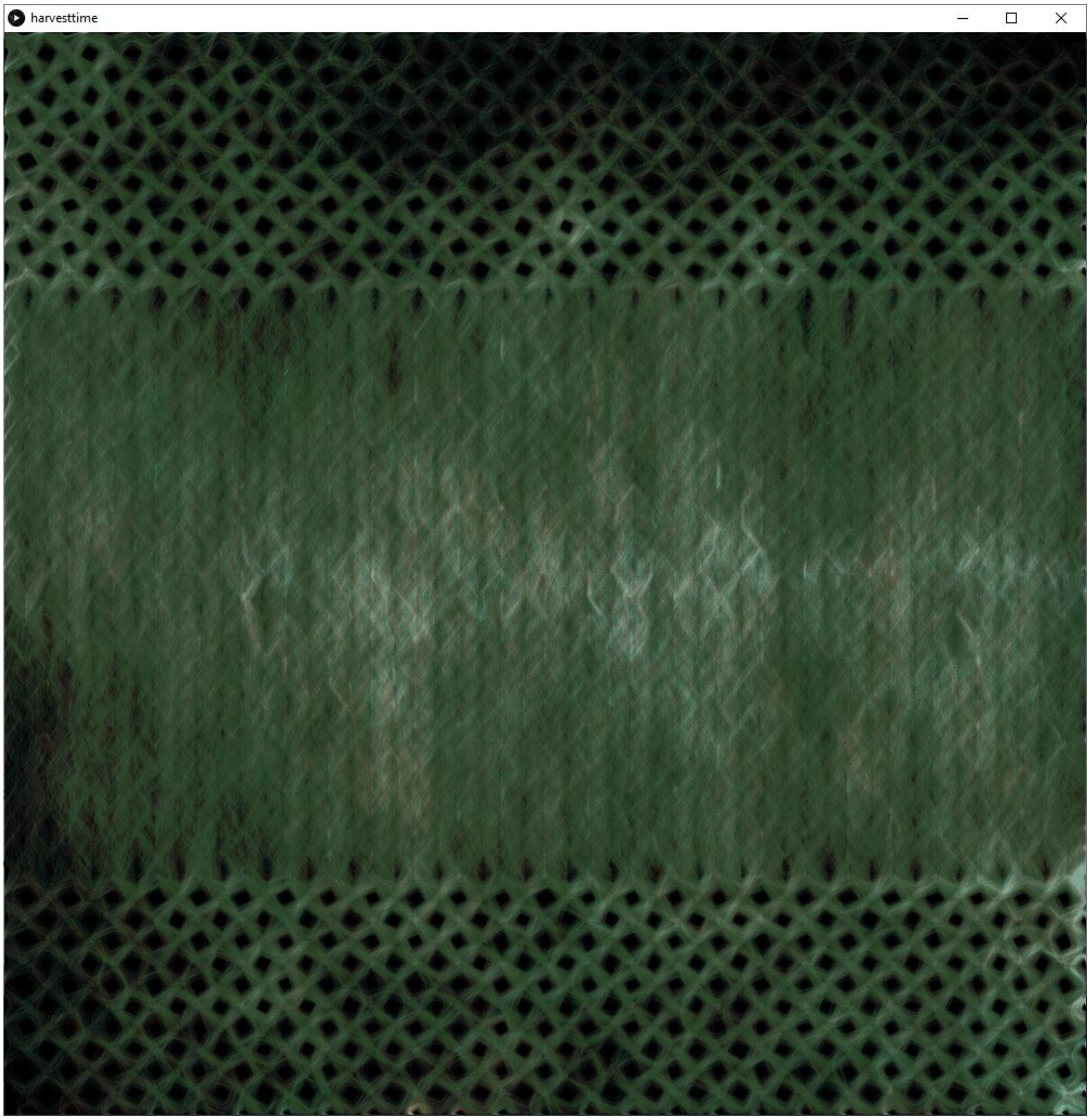
Perlin Noise is a type of noise that has been developed by Ken Perlin. What special about it is, the noise created this way have smooth transitions between elements. Every element of the noise array has very minimal difference. This smoothness can be used in computer graphics. In fact, Ken Perlin developed this technique to make CGIs (Computer Generated Imaginary) more smooth. The following generative arts are also used the Perlin Noise to have more organic sight.

Generative Arts Samples:

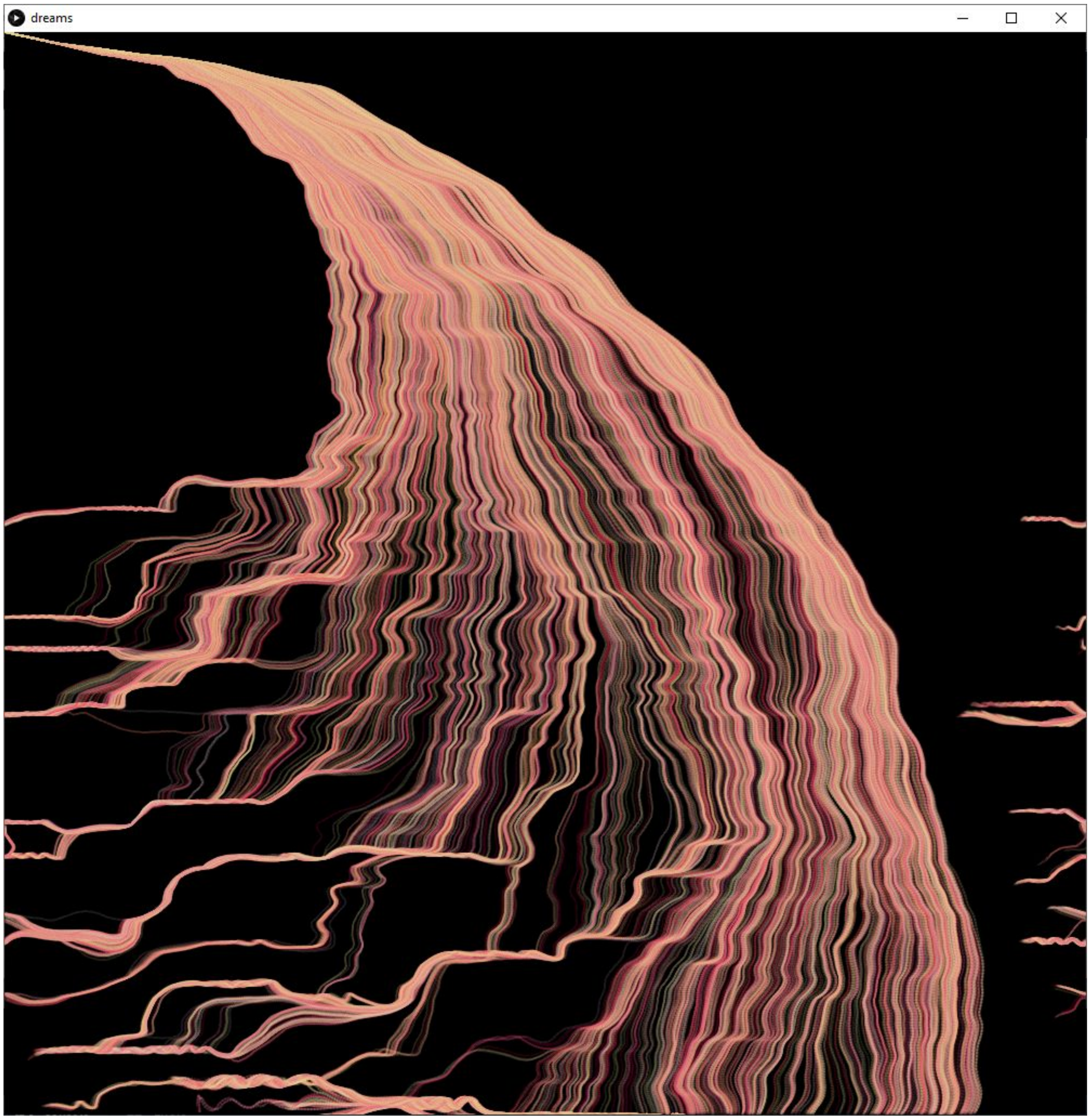
Title: Next Dimension



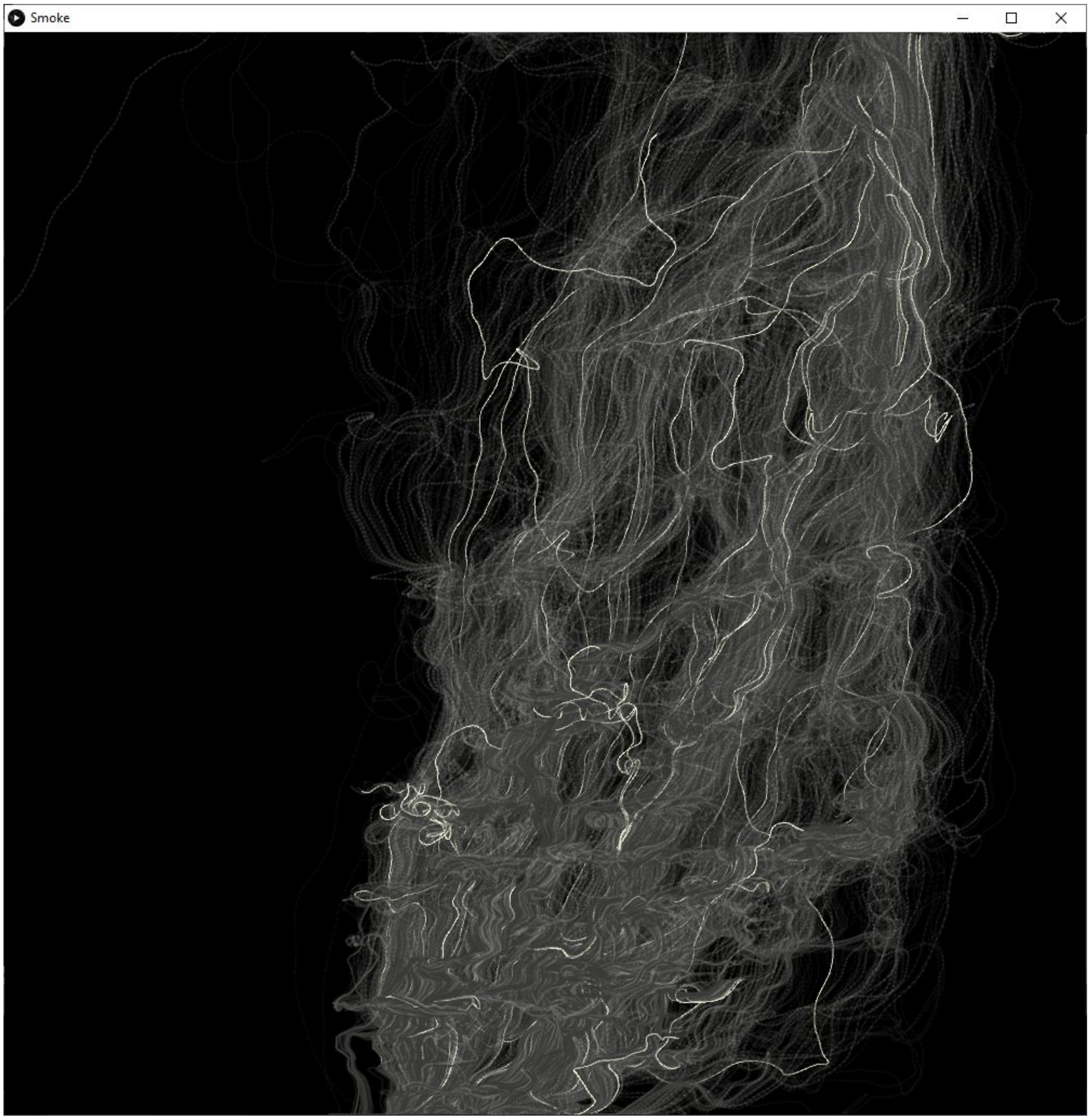
Title : Harvest Time



Title : Dreams



Title : Smoke



Code Structure:

Next Dimension:

Particle:

```
public class Particle {
  PVector pos;
  PVector vel;
  PVector acc;
  PVector previousPos;
  float maxSpeed;
  int r, g, b;

  Particle(PVector start, float maxspeed, int pr, int pg, int pb) {
    maxSpeed = maxspeed;
    pos = start;
    vel = new PVector(0, 0);
    acc = new PVector(0, 0);
    previousPos = pos.copy();
    r = pr;
    g = pg;
    b = pb;
  }

  void run() {
    updatePosition();
    edges();
    show();
  }

  void updatePosition() {
    pos.add(vel);
    vel.add(acc);
    vel.limit(maxSpeed);
    acc.mult(0);
  }

  void applyForce(PVector force) {
    acc.add(force);
  }
  void show() {
    stroke(r, g, b, 5);
    //strokeWeight(3);
    strokeWeight(1);
    line(pos.x, pos.y, previousPos.x, previousPos.y);
    //point(pos.x, pos.y);
    updatePreviousPos();
  }

  void updatePreviousPos() {
    this.previousPos.x = pos.x;
    this.previousPos.y = pos.y;
  }

  void edges() {
    if (pos.x > width) {
      pos.x = 0;
      updatePreviousPos();
    }
    if (pos.x < 0) {
      pos.x = width;
      updatePreviousPos();
    }
    if (pos.y > height) {
      pos.y = 0;
      updatePreviousPos();
    }
    if (pos.y < 0) {
      pos.y = height;
      updatePreviousPos();
    }
  }

}
```

Fixed Attractor:

```
class FixedAttractor{
  PVector position;
  float mass;



FixedAttractor(float xpos, float ypos, float amass){
  mass = amass;
  position = new PVector(xpos, ypos);
}


PVector getAttractionForceOn(Particle mp){
  PVector attrForce = PVector.sub(position, mp.pos);

  float distance = attrForce.mag();

  // F_g Magnitude
  float F_g_mag = (mass) / (distance*distance);

  attrForce.setMag(F_g_mag);

  return attrForce;
}


void display() {
    stroke(0);
    fill(175);
    ellipse(position.x, position.y, 2*sqrt(mass), 2*sqrt(mass));
  }


}
```

Next Dimension:

```
ArrayList<Particle> particles;
FixedAttractor fa1;

boolean debug = false;
int step = 0;

float f1offx = 0;

float f1offy = -5;

void setup() {
  size(1000, 1000, P2D);
  fa1 = new FixedAttractor(width/2, height/2, 5000);
  initNParticles(5000);

  background(0);
}

void draw() {
  step ++;



  if (step%10 == 0){

    f1offx += 0.01;
    f1offy -= 0.01;
    fa1.position = new PVector(noise(f1offx)*width, noise(f1offy)*height);

  }



  for (Particle p : particles) {
    PVector attraction_force1 = fa1.getAttractionForceOn(p);
    p.applyForce(attraction_force1);
    p.run();
  }
}

// This is how we init particles
void initNParticles(int n){
  particles = new ArrayList<Particle>();
  for (int i = 0; i < n; i++) {
    float maxSpeed = random(1, 1);


    if (i%4 == 0){
      PVector start_point = new PVector(random(width), 0);
      particles.add(new Particle(start_point, maxSpeed, 100, 50, 200));
    }
    else if (i%4 == 1){
      PVector start_point = new PVector(0, random(height));
      particles.add(new Particle(start_point, maxSpeed, 200, 0, 100));
    }
    else if (i%4 == 2){
      PVector start_point = new PVector(width, random(height));
      particles.add(new Particle(start_point, maxSpeed, 200, 0, 100));
    }
    else{
      PVector start_point = new PVector(random(width), height);
      particles.add(new Particle(start_point, maxSpeed, 100, 50, 200));
    }

  }
}
```

Harvest Time:

Particle

```
public class Particle {
  PVector pos;
  PVector vel;
  PVector acc;
  PVector previousPos;
  float maxSpeed;
  int r, g, b;

  int step = 0;

  Particle(PVector start, float maxspeed, int pr, int pg, int pb) {
    maxSpeed = maxspeed;
    pos = start;
    vel = new PVector(0, 0);
    acc = new PVector(0, 0);
    previousPos = pos.copy();
    r = pr;
    g = pg;
    b = pb;
  }

  void run() {
    step += 0.1 ;
    updatePosition();
    edges();
    show();
  }

  void updatePosition() {
```

```
  pos.add(vel);
  vel.add(acc);
  vel.limit(maxSpeed);
  acc.mult(0);
}

void applyForce(PVector force) {
  acc.add(force);
}

void applyFieldForce(FlowField flowfield) {
  int x = floor(pos.x / flowfield.scl);
  int y = floor(pos.y / flowfield.scl);

  PVector force = flowfield.vectors[x][y];
  applyForce(force);
}

void show() {
  stroke(r, g, b, 2);
  strokeWeight(1);
  line(pos.x, pos.y, previousPos.x, previousPos.y);
  //point(pos.x, pos.y);
  updatePreviousPos();
}

void updatePreviousPos() {
  this.previousPos.x = pos.x;
  this.previousPos.y = pos.y;
}

void edges() {
  if (pos.x > width) {
    pos.x = width;
    updatePreviousPos();
  }
  if (pos.x < 0) {
    pos.x = 0;
    updatePreviousPos();
  }
  if (pos.y > height) {
    pos.y = height;
    updatePreviousPos();
  }
```

```
    if (pos.y < 0) {
      pos.y = 0;
      updatePreviousPos();
    }
  }

}
```

FlowField:

```
public class FlowField {
  PVector[][] vectors;
  int cols, rows;
  float grid_inc = 0.1;
  float noise_time_off = 0;
  float noise_time_inc = 0.002;
  int scl;


  FlowField(int res, float ntime_inc) {
    scl = res;
    noise_time_inc = ntime_inc;
    cols = floor(width / res) + 1;
    rows = floor(height / res) + 1;
    vectors = new PVector[cols] [rows];
  }

  void updateFF() {
    float xoff = 0;
    for (int y = 0; y < rows; y++) {
      float yoff = 0;
      for (int x = 0; x < cols; x++) {
        float angle = noise(xoff, yoff, noise_time_off) * TWO_PI * 4 ;

        PVector v = PVector.fromAngle(angle);

        //v.add(new PVector(0.1, -0.1));

       if (x % 2 == 0){
         v.add(0, -1.5);
       }else{
         v.add(0, 1.5);
       }
       if ((y<rows/4 || y>3*rows/4) && y % 2 == 0){
         v.add(-1.5, 0);
       }else if ((y<rows/4 || y>3*rows/4) && y % 2 == 1){
         v.add(1.5, 0);
       }


        v.setMag(1);

        vectors[x][y] = v;

        xoff += grid_inc;
      }
      yoff += grid_inc;
    }
    noise_time_off += noise_time_inc;
  }


  void display() {
    for (int y = 0; y < rows; y++) {
      for (int x = 0; x < cols; x++) {
        PVector v = vectors[x][y];

        stroke(0, 0, 0, 150);
        strokeWeight(1);
        pushMatrix();
        translate(x * scl, y * scl);
        rotate(v.heading());
        line(0, 0, scl, 0);
        popMatrix();
      }
    }
  }
}
```

Harvest Time:

```
FlowField flowfield;
ArrayList<Particle> particles;

boolean debug = false;

void setup() {
  size(1000, 1000, P2D);
  flowfield = new FlowField(20, 0.01);
  flowfield.updateFF();

  initNParticles(10000);

  background(0);
}

void draw() {
  //background(0); // Comment out to see the effect
  flowfield.updateFF();

  if (debug) flowfield.display();


  for (Particle p : particles) {
    p.applyFieldForce(flowfield);
    p.run();
  }

}

// This is how we init particles
void initNParticles(int n){
  particles = new ArrayList<Particle>();
  for (int i = 0; i < n; i++) {
    float maxSpeed = random(1, 2);

    if(i%2 == 0){
      PVector start_point = new PVector(random(width), height/2);
      particles.add(new Particle(start_point, maxSpeed, 227, 100, 100));
    }
    else{
      PVector start_point = new PVector(random(width), height/2);
      particles.add(new Particle(start_point, maxSpeed, 66, 255, 211));
    }

  }
}
```

Dreams

Particle:

```
public class Particle {
  PVector pos;
  PVector vel;
  PVector acc;
  PVector previousPos;
  float maxSpeed;
  int r, g, b;

  Particle(PVector start, float maxspeed, int pr, int pg, int pb) {
    maxSpeed = maxspeed;
    pos = start;
    vel = new PVector(0, 0);
    acc = new PVector(0, 0);
    previousPos = pos.copy();
    r = pr;
    g = pg;
    b = pb;
  }

  void run() {
    updatePosition();
    edges();
    show();
  }

  void updatePosition() {
    pos.add(vel);
    vel.add(acc);
    vel.limit(maxSpeed);
    acc.mult(0);
  }

  void applyForce(PVector force) {
    acc.add(force);
  }

  void applyFieldForce(FlowField flowfield) {
    int x = floor(pos.x / flowfield.scl);
    int y = floor(pos.y / flowfield.scl);
    PVector force = flowfield.vectors[x][y];
    applyForce(force);
  }

  void show() {
    stroke(r, g, b, 20);
    //strokeWeight(3);
    strokeWeight(2);
    line(pos.x, pos.y, previousPos.x, previousPos.y);
    //point(pos.x, pos.y);
    updatePreviousPos();
  }

  void updatePreviousPos() {
    this.previousPos.x = pos.x;
    this.previousPos.y = pos.y;
  }

  void edges() {
    if (pos.x > width) {
      pos.x = 0;
      updatePreviousPos();
    }
    if (pos.x < 0) {
      pos.x = width;
      updatePreviousPos();
    }
    if (pos.y > height) {
      pos.y = height;
      updatePreviousPos();
    }
    if (pos.y < 0) {
      pos.y = 0;
      updatePreviousPos();
    }
  }

}
```

FlowField:

```
public class FlowField {
  PVector[][] vectors;
```

```
int cols, rows;
float grid_inc = 0.1;
float noise_time_off = 0;
float noise_time_inc = 0.002;
int scl;

int step = 1;

FlowField(int res, float ntime_inc) {
  scl = res;
  noise_time_inc = ntime_inc;
  cols = floor(width / res) + 1;
  rows = floor(height / res) + 1;
  vectors = new PVector[cols] [rows];
}

void updateFF() {
  float xoff = 0;
  for (int y = 0; y < rows; y++) {
    float yoff = 0;
    for (int x = 0; x < cols; x++) {
      float angle = noise(xoff, yoff, noise_time_off) * TWO_PI * 0.05*step;

      PVector v = PVector.fromAngle(angle);
      v.setMag(1);
      vectors[x][y] = v;

      xoff += grid_inc;
    }
    yoff += grid_inc;
  }
  noise_time_off += noise_time_inc;
  step ++;
  if(step >= 40){
    step =1;
  }
}


void display() {
  for (int y = 0; y < rows; y++) {
    for (int x = 0; x < cols; x++) {
      PVector v = vectors[x][y];
```

```
        stroke(0, 0, 0, 150);
        strokeWeight(1);
        pushMatrix();
        translate(x * scl, y * scl);
        rotate(v.heading());
        line(0, 0, scl, 0);
        popMatrix();
      }
    }
  }
}
```

Dreams:

```
FlowField flowfield;
ArrayList<Particle> particles;

boolean debug = false;


int step = 0;

void setup() {
  size(1000, 1000, P2D);

  flowfield = new FlowField(10, 0.009);
  flowfield.updateFF();

  initNParticles(2500);

  background(0);
}

void draw() {
  step ++;

  if (step%50 == 0){
    flowfield.updateFF();
  }

  if (debug) flowfield.display();

  for (Particle p : particles) {
    p.applyFieldForce(flowfield);
    p.run();
  }
}

// This is how we init particles
void initNParticles(int n){
  particles = new ArrayList<Particle>();
  for (int i = 0; i < n; i++) {
    float maxSpeed = random(1, 3);

    if (i < n*(0.33)){
      PVector start_point = new PVector(0, 0);
      particles.add(new Particle(start_point, maxSpeed, 247, 15, 81));
    }
    else if(i < n*(0.66)){
      PVector start_point = new PVector(0, 0);
      particles.add(new Particle(start_point, maxSpeed, 209, 180, 188));
    }
    else{
      PVector start_point = new PVector(0, 0);
      particles.add(new Particle(start_point, maxSpeed, 237, 226, 126));
    }

  }
}
```

Smoke

Particle:

```
public class Particle {
  PVector pos;
  PVector vel;
  PVector acc;
  PVector previousPos;
  float maxSpeed;
  int r, g, b;

  Particle(PVector start, float maxspeed, int pr, int pg, int pb) {
    maxSpeed = maxspeed;
    pos = start;
    vel = new PVector(0, 0);
    acc = new PVector(0, 0);
    previousPos = pos.copy();
    r = pr;
    g = pg;
    b = pb;
  }

  void run() {
    updatePosition();
    edges();
    show();
  }

  void updatePosition() {
    pos.add(vel);
    vel.add(acc);
    vel.limit(maxSpeed);
    acc.mult(0);
  }

  void applyForce(PVector force) {
    acc.add(force);
  }

  void applyFieldForce(FlowField flowfield) {
    int x = floor(pos.x / flowfield.scl);
    int y = floor(pos.y / flowfield.scl);

    PVector force = flowfield.vectors[x][y];
    applyForce(force);
  }

  void show() {
    if (r == 249){
      stroke(r, g, b, 255);
      strokeWeight(1);
    }
    else{
    stroke(r, g, b, 25);
    strokeWeight(2);
    }
    //strokeWeight(2);
    line(pos.x, pos.y, previousPos.x, previousPos.y);
```

```
    point(pos.x, pos.y);
    updatePreviousPos();
  }

  void updatePreviousPos() {
    this.previousPos.x = pos.x;
    this.previousPos.y = pos.y;
  }

  void edges() {
    if (pos.x > width) {
      pos.x = 0;
      updatePreviousPos();
    }
    if (pos.x < 0) {
      pos.x = width;
      updatePreviousPos();
    }
    if (pos.y > height) {
      pos.y = height;
      updatePreviousPos();
    }
    if (pos.y < 0) {
      pos.y = 0;
      updatePreviousPos();
    }
  }

}
```

Flowfield:

```
public class FlowField {
  PVector[][] vectors;
  int cols, rows;
  float grid_inc = 0.1;
  float noise_time_off = 0;
  float noise_time_inc = 0.002;
  int scl;

  FlowField(int res, float ntime_inc) {
    scl = res;
    noise_time_inc = ntime_inc;
    cols = floor(width / res) + 1;
    rows = floor(height / res) + 1;
  vectors = new PVector[cols] [rows];
}

void updateFF() {
  float xoff = 0;
  for (int y = 0; y < rows; y++) {
    float yoff = 0;
    for (int x = 0; x < cols; x++) {
      float angle = noise(xoff, yoff, noise_time_off) * TWO_PI * 2;


      PVector v = PVector.fromAngle(angle);

      v.add(new PVector(0, -0.75));


      if(x<cols/5){
        v.add(new PVector(0.6, 0));
      }else if (x > 4*cols/5){
        v.add(new PVector(-0.6, 0));
      }


      v.setMag(1);
      vectors[x][y] = v;

      xoff += grid_inc;
    }
    yoff += grid_inc;
  }
  noise_time_off += noise_time_inc;
}


void display() {
  for (int y = 0; y < rows; y++) {
    for (int x = 0; x < cols; x++) {
      PVector v = vectors[x][y];

      stroke(0, 0, 0, 150);
      strokeWeight(1);
      pushMatrix();
      translate(x * scl, y * scl);
      rotate(v.heading());
```

```
        line(0, 0, scl, 0);
        popMatrix();
      }
    }
  }
}
```

Smoke:

```
FlowField flowfield;
ArrayList<Particle> particles;

boolean debug = false;

void setup() {
  size(1000, 1000, P2D);

  flowfield = new FlowField(10, 0.009);
  flowfield.updateFF();

  initNParticles(4000);

  background(0);
}

void draw() {
  flowfield.updateFF();

  if (debug) flowfield.display();

  for (Particle p : particles) {
    p.applyFieldForce(flowfield);
    p.run();
  }
}

// This is how we init particles
void initNParticles(int n){
  particles = new ArrayList<Particle>();
  for (int i = 0; i < n; i++) {
    float maxSpeed = random(1, 7);

    if (i <= n*(0.20)){
      PVector start_point = new PVector(random(width/5)+width/2-width/5, height);
      particles.add(new Particle(start_point, maxSpeed, 155, 156, 152));
    }
    else if (i <= n*(0.21)){
      PVector start_point = new PVector(random(width/5)+width/2-width/5, height);
      particles.add(new Particle(start_point, maxSpeed, 249, 250, 222));
    }
    else{
      PVector start_point = new PVector(random(width/5)+width/2-width/5, height);
      particles.add(new Particle(start_point, maxSpeed, 53, 54, 52));
    }

  }
}
```

Parameters:

Next Dimension:

- # of particles = 5000
- Position of the attractor is updated in every 10 invocations of the draw function. (see line 30)
- Particles are placed at the boundaries.

Harvest Time:

- # of particles = 10000
- All the particles are initialized at the middle of the screen horizontally.
- Bias vector in the flowfield has the magnitude of 1.5

Dreams:

- # of particles = 2500
- All initialized at 0,0
- The angle of the each flow field vector oscillates between 0 an 4PI

Smoke:

- # of particles = 4000
- Particles are initialized at the center region of the bottom.
- There is a bias in the flow field such that every particle moves upward and they are pushed to the center if they cross to the first or last 20% of the width. (see FlowField line 30)