0. Introduction

박종화
suakii@gmail.com

0.1 Random Walks

Flip 1	Flip 2	Result
Heads	Heads	Step forward.
Heads	Tails	Step right.
Tails	Heads	Step left.
Tails	Tails	Step backward.

기체 내부의 입자운동
카니노에서 사람들이 돈을 소비하는 행동 방식
현실 세계에서의 여러 가지 현상의 간단한 모델링이 가능.

0.2 Walker Class

- Class
  - 객체를 만드는데 사용되는 템플릿
- Object
  - 속성(변수)과 행동 양식(함수)를 갖는 실체

0.2 Walker Class

```
//객체는 데이터를 가짐
class Walker {
    int x;
    int y;
```

```
//생성자-객체가 생성될 때 자
//동으로 호출되는 함수
Walker() {
        x = width/2;
        y = height/2;
}
```

0.2 Walker Class

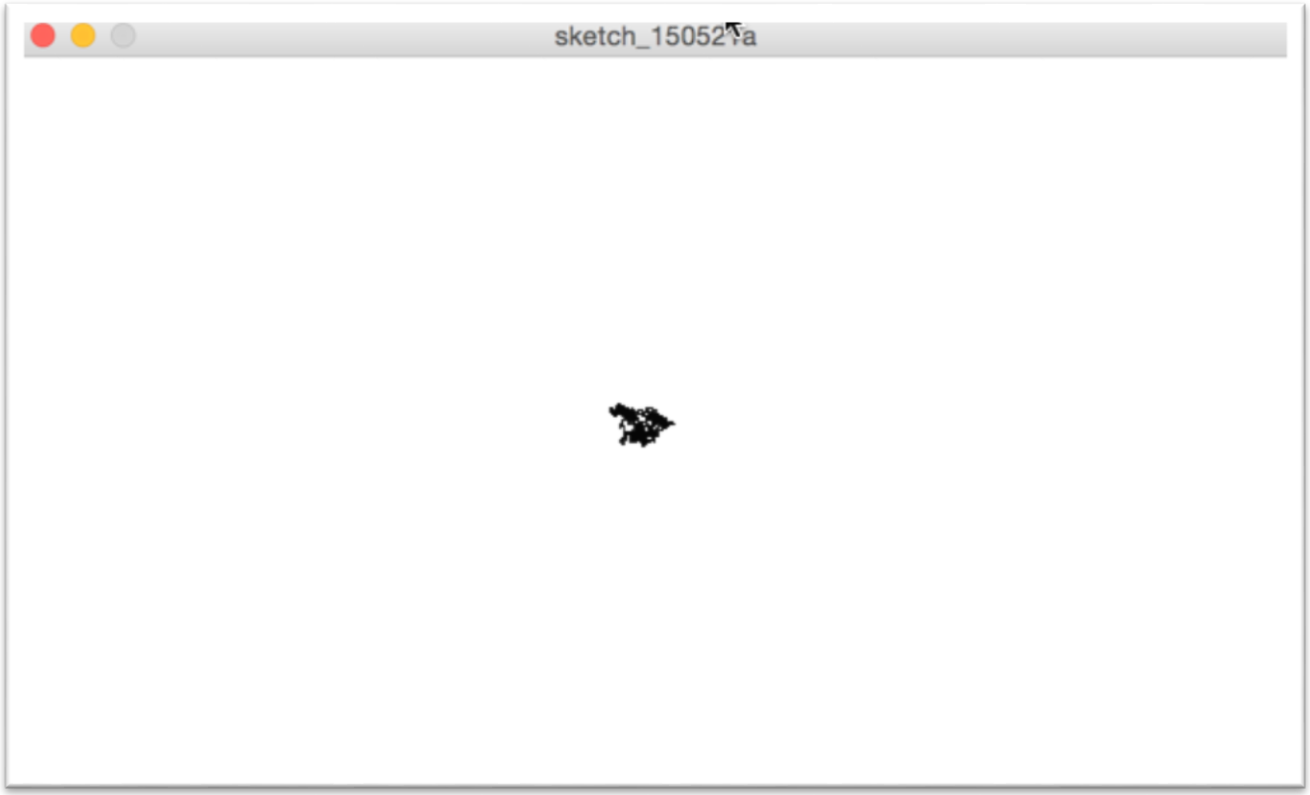
```
//기능(함수)을 추가할 수 있음.
void display() {
        stroke(0);
        point(x,y);
}
```

```
//이동 함수 추가
void step() {
  int choice = int(random(4));
  if (choice == 0) {
     x++;
  } else if (choice == 1) {
     x--;
  } else if (choice == 2) {
    y++;
  } else {
    y--;
  }
}
```

0.2 Walker Class

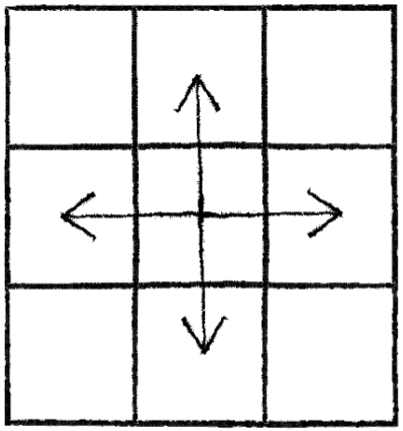
```
Walker w;//전역 위치

void setup() {
  size(800,200);
  // Create a walker object
  w = new Walker();
  background(255);
}

void draw() {
  // Run the walker object
  w.step();
  w.display();
}
```

0.2 Walker Class



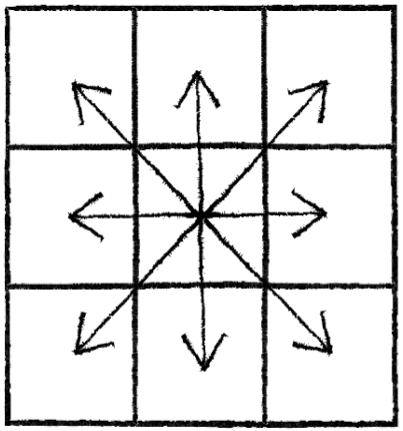
0.2 Walker Class

4 possible steps



4 possible steps

8 possible steps



8 possible steps

0.2 Walker Class

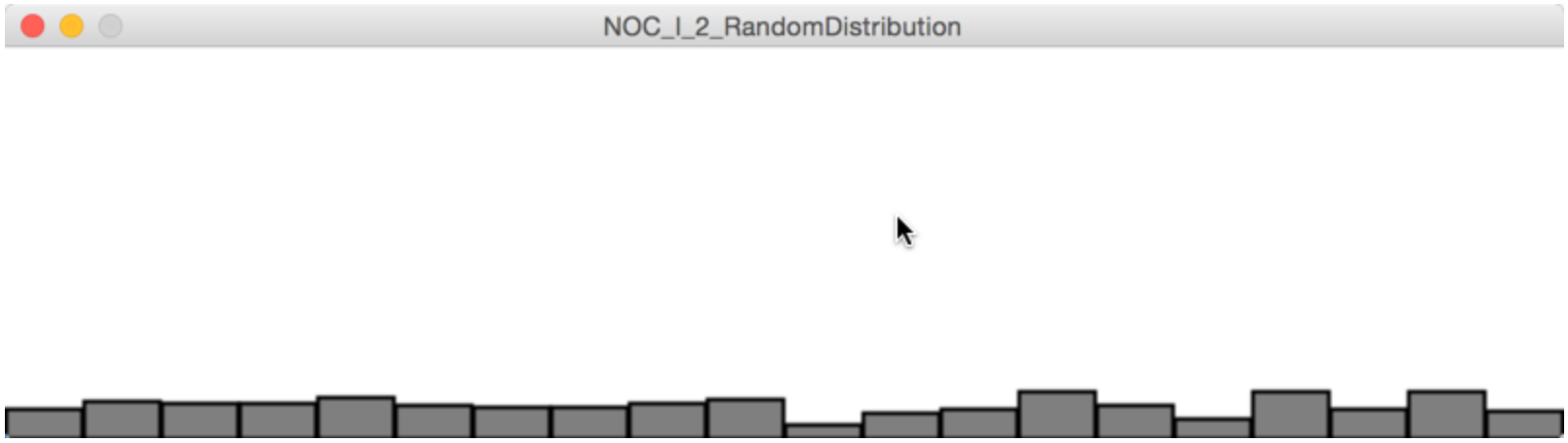
```
void step() {
  int stepx = int(random(3))-1;
  int stepy = int(random(3))-1;
  x += stepx;
  y += stepy;
}
//25%의  확률
```

```
void step() {
  float stepx = random(-1, 1);
  float stepy = random(-1, 1);
  x += stepx;
  y += stepy;
}
//x, y float
//11.1%의 확률
```

0.2 Random number distribution

```
// An array to keep track of how often random numbers are picked
float[] randomCounts;
void setup() {
  size(800,200);
  randomCounts = new float[20];
}

void draw() {
  background(255);
  // Pick a random number and increase the count
  int index = int(random(randomCounts.length));
  randomCounts[index]++;
  // Draw a rectangle to graph results
  stroke(0);
  strokeWeight(2);
  fill(127);

  int w = width/randomCounts.length;

  for (int x = 0; x < randomCounts.length; x++) {
    rect(x*w,height-randomCounts[x],w-1,randomCounts[x]);
  }
}
```

0.2 Random number distribution



0.3 Probability and Non-Uniform Distributions

```
int[] stuff = new int[5]
stuff[0] = 1;
stuff[1] = 1;
stuff[2] = 2;
stuff[3] = 3;
stuff[4] = 3;
int index = int(random(stuff.length));
/*
1 = 40%
2 = 20%
3 = 40%
*/
```

0.3 Probability and Non-Uniform Distributions

- between 0.00 and 0.60 (60%) –> Outcome A
- between 0.60 and 0.70 (10%) –> Outcome B
- between 0.70 and 1.00 (30%) –> Outcome C

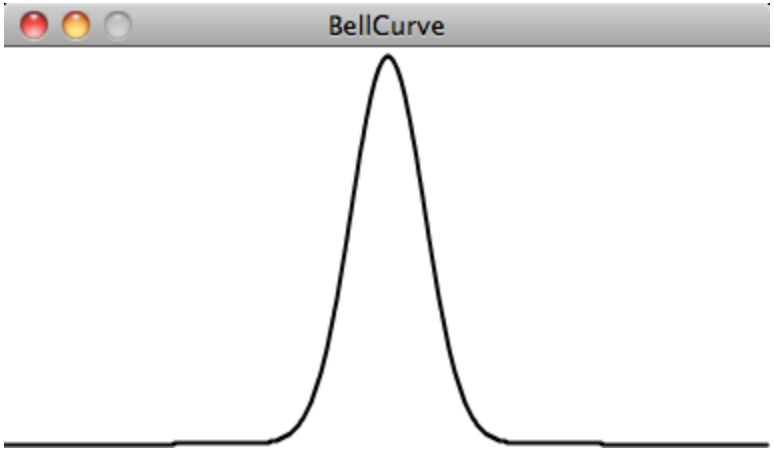
```
float num = random(1);
if (num < 0.6) {
        println("Outcome A");
} else if (num < 0.7) {
        println("Outcome B");
} else {
        println("Outcome C");
}
```

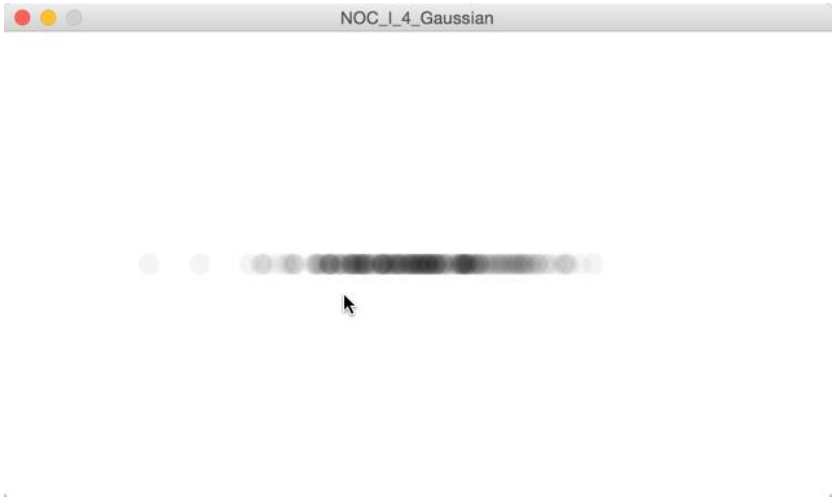
0.3 Probability and Non-Uniform Distributions

- Walker Class below probabilities.
  - chance of moving up: 20%
  - chance of moving down: 20%
  - chance of moving left: 20%
  - chance of moving right: 40%

```
void step() {
        float r = random(1);
        //A 40% chance of moving to the right!
        if (r < 0.4) {
                x++;
        } else if (r < 0.6) {
                x--;
        } else if (r < 0.8) {
                y++;
        } else { y--;
        }
}
```

0.4 Normal Distribution of Random Numbers





0.4 Normal Distribution of Random Numbers

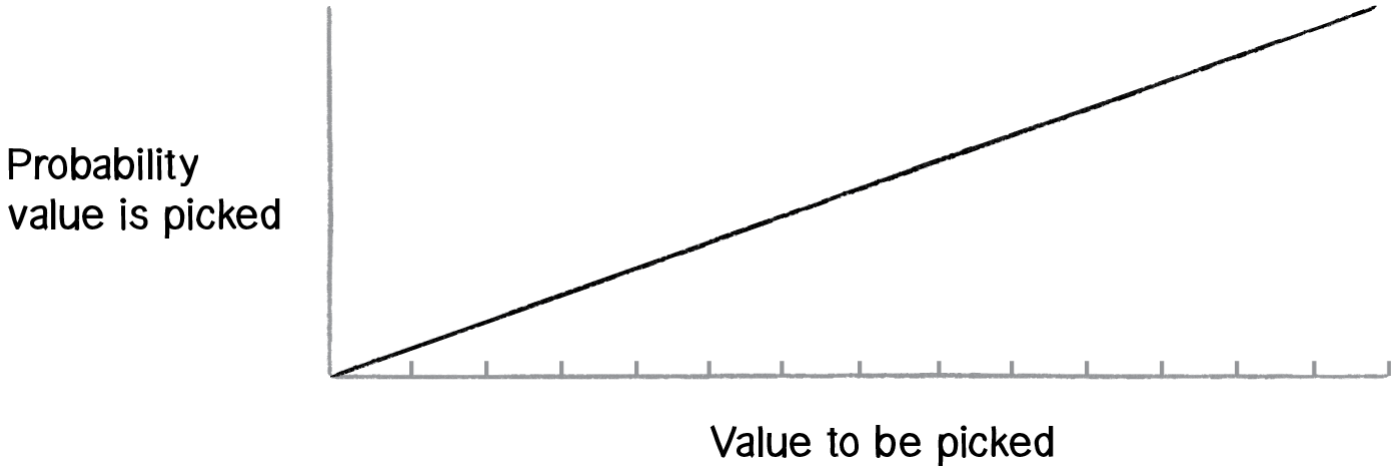
```
void setup() {
  size(640, 360);
  background(255);
}

void draw() {

  // Get a gaussian random number w/ mean of 0 and standard deviation of 1.0
  float xloc = randomGaussian();

  float sd = 60;                  // Define a standard deviation
  float mean = width/2;           // Define a mean value (middle of the screen along the x-axis)
  xloc = ( xloc * sd ) + mean;  // Scale the gaussian random number by standard deviation and mean

  noStroke();
  fill(0, 10);
  noStroke();
  ellipse(xloc, height/2, 16, 16);   // Draw an ellipse at our "normal" random location
}
```

0.5 A Custom Distribution of Random Numbers

```
float r = random(1);
A 1% chance of taking a large step
if (r < 0.01) {
        xstep = random(-100,100);
        ystep = random(-100,100);
 } else {
        xstep = random(-1,1);
        ystep = random(-1,1);
}
```

- **We could implement a Levy flight by saying that there is a 1% change of the walker taking a large step.**

0.5 A Custom Distribution of Random Numbers



x가 임의의 숫자일 때 y = x 라는 함수를 만들어 y를 확률로 사용할 수 있을까?
즉, 큰 숫자가 나올 수록 사용될 확률을 높이는 것이다.

0.5 A Custom Distribution of Random Numbers

- Pick a random number: R1
- Compute a probability P that R1 should qualify. Let's try: P = R1.
- Pick another random number: R2
- If R2 is less than P, then we have found our number—R1!
- If R2 is not less than P, go back to step 1 and start over.

```
float montecarlo() {
        //We do this "forever" until we find a qualifying random value.
        while (true) {
                //Pick a random value.
                float r1 = random(1);
                //Assign a probability.
                float probability = r1;
                //Pick a second random value.
                float r2 = random(1);
                //Does it qualify? If so, we're done!
                if (r2 < probability) {
                        return r1;
                }
        }
}
```

0.5 A Custom Distribution of Random Numbers

Walker

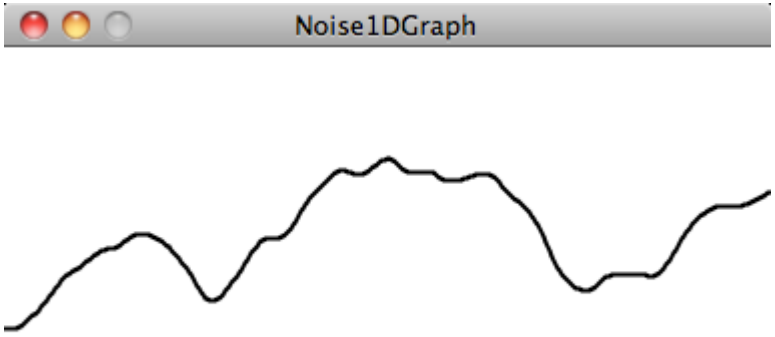
```
class Walker {
  float x, y;

  float prevX, prevY;

  Walker() {
    x = width/2;
    y = height/2;
  }

  void render() {
    stroke(255);
    line(prevX,prevY,x, y);
  }

  // Randomly move according to floating point values
  void step() {
    prevX = x;
    prevY = y;

    float stepx = random(-1, 1);
    float stepy = random(-1, 1);

    float stepsize = montecarlo()*50;
    stepx *= stepsize;
    stepy *= stepsize;

    x += stepx;
    y += stepy;
    x = constrain(x, 0, width-1);
    y = constrain(y, 0, height-1);
  }
}
```

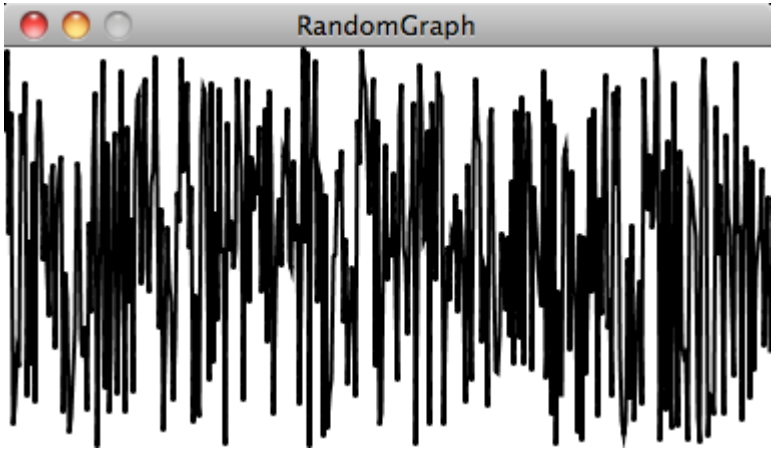
Can you map the probability exponentially—i.e. making the likelihood that a value is picked equal to the value squared?

```
float montecarlo() {
  while (true) {

    float r1 = random(1);
    float probability = pow(1.0 - r1,8);

    float r2 = random(1);
    if (r2 < probability) {
        return r1;
    }
  }
}
```

0.6 Perlin Noise(A Smoother Approach)

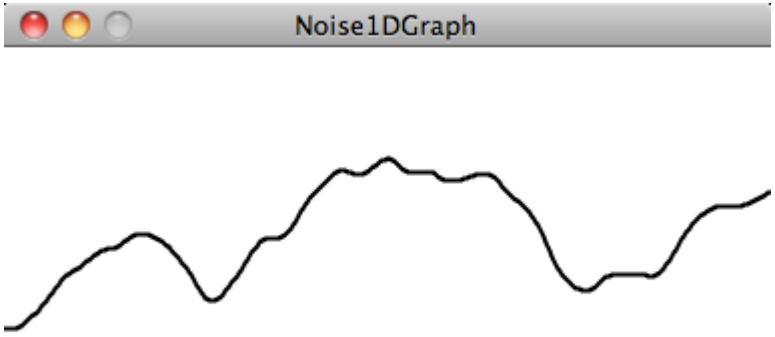
A good random number generator produces numbers that have no relationship and show no discernible pattern.

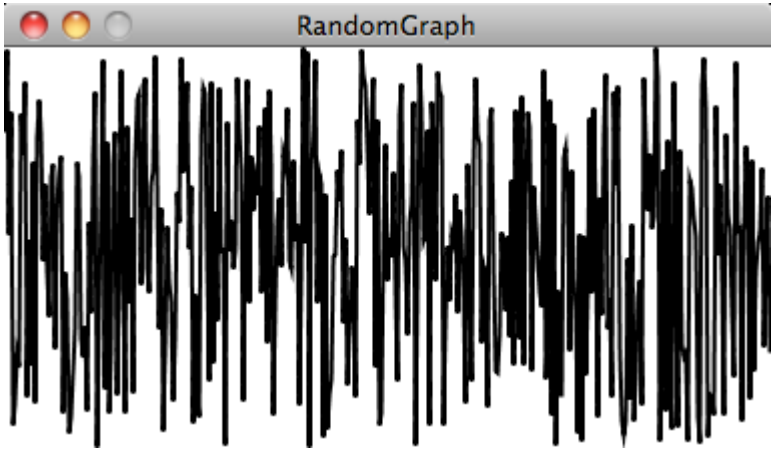




0.6 Perlin Noise(A Smoother Approach)

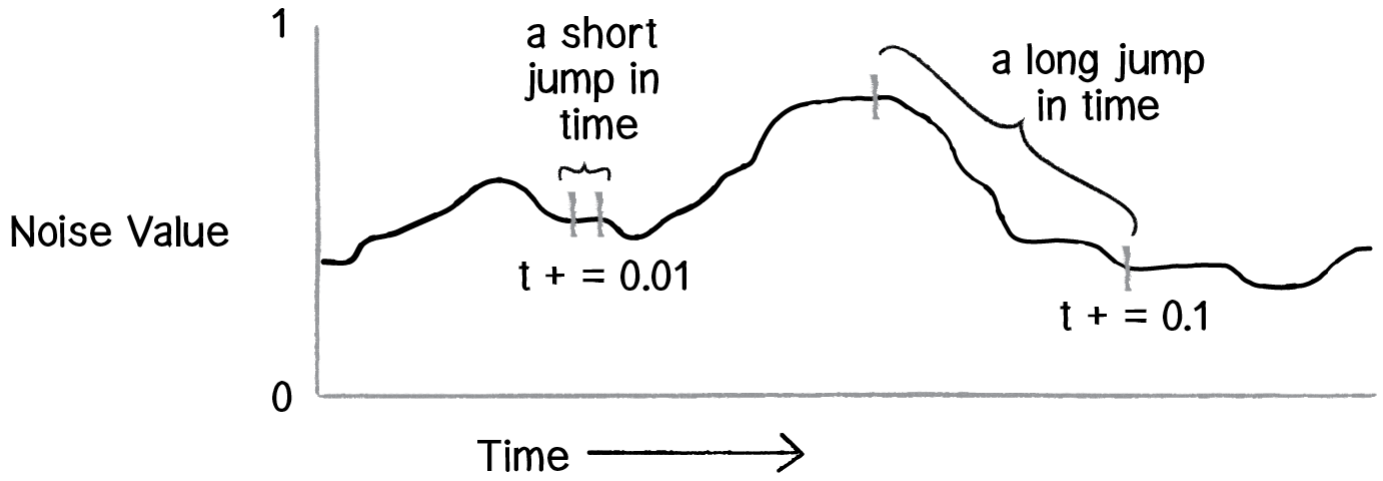
A good random number generator produces numbers that have no relationship and show no discernible pattern.



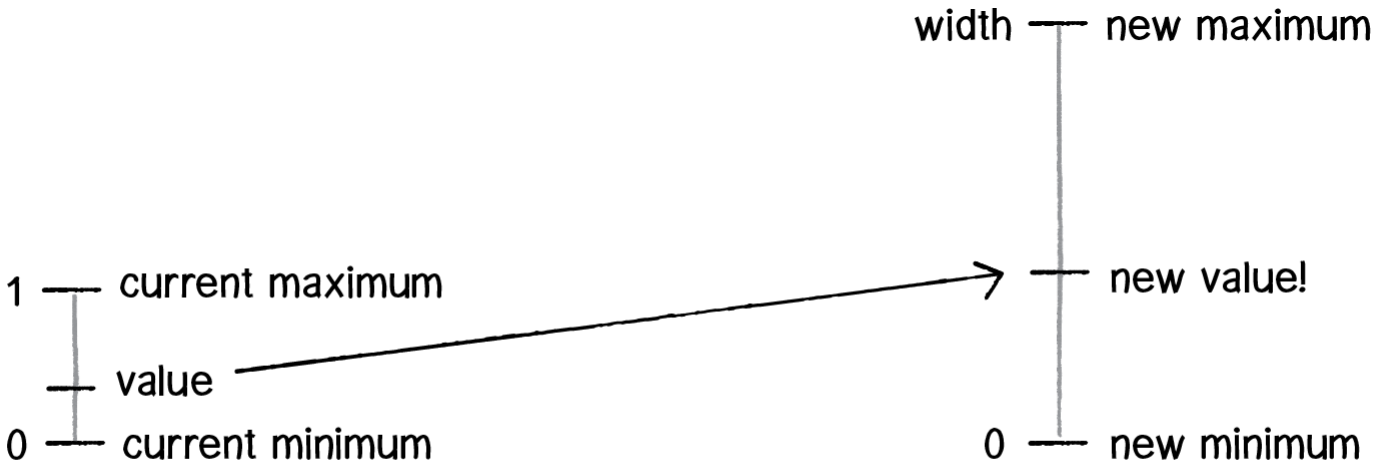


0.6 Perlin Noise(A Smoother Approach)

Perlin noise is a random sequence generator producing a more natural, harmonic succession of numbers than that of the standard random()function. It always returns a value between 0 and 1.



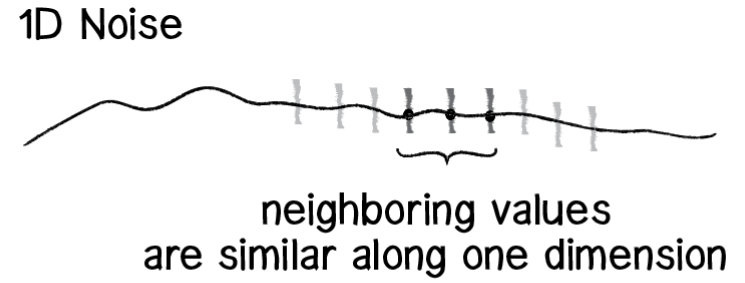
0.6.1 Mapping Noise

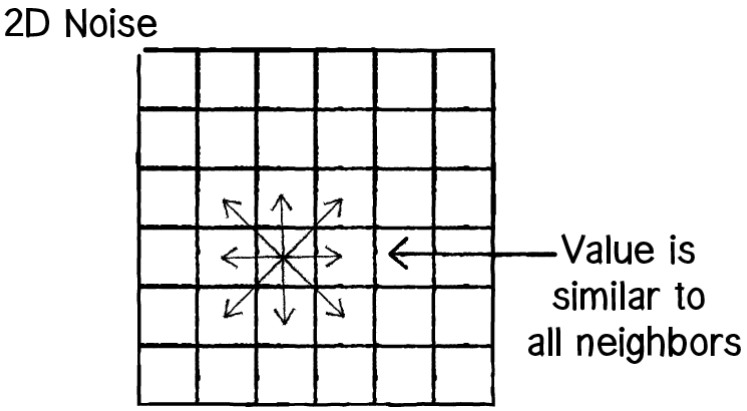


new value= map(value, current min, current max, new min, new max)

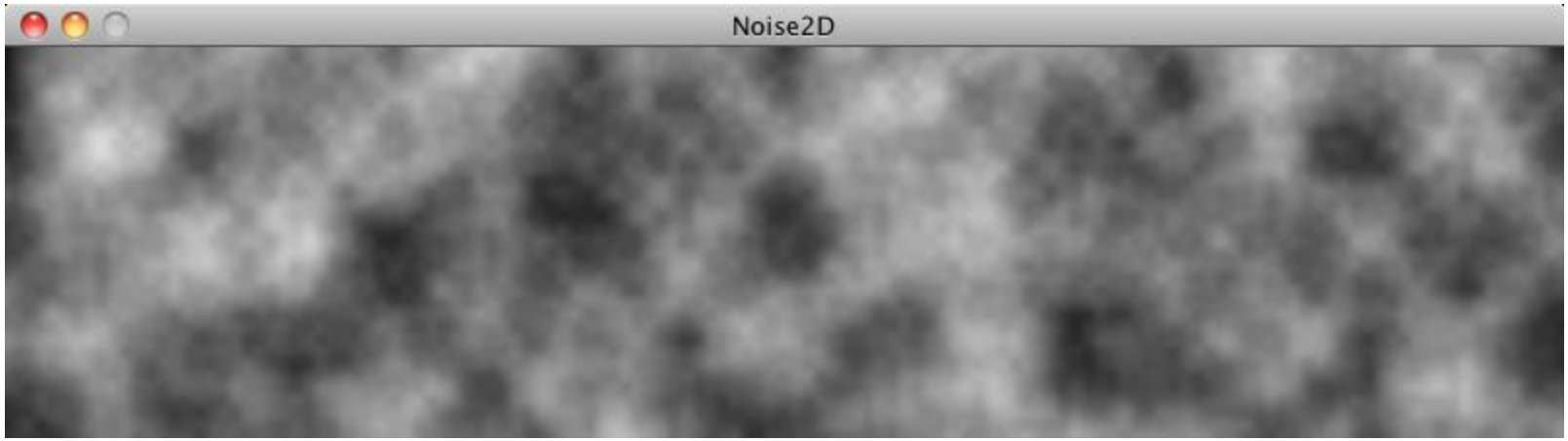
```
float t = 0;
void setup() {
  size(800,600);
}
void draw() {
  float n = noise(t);
  float x = map(n,0,1,0,width);
  ellipse(x,180,16,16);

  t += 0.01;
}
```

0.6.2 Two-Dimensional Noise



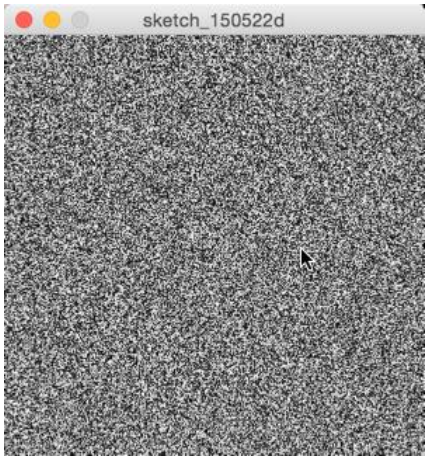


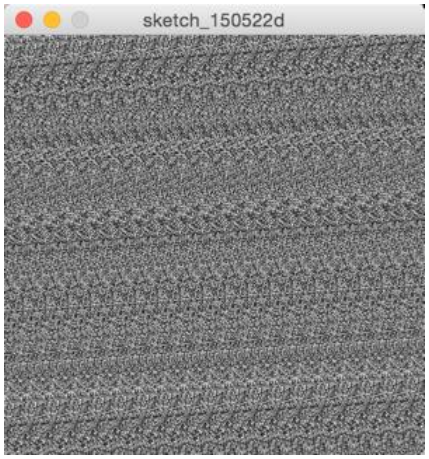
0.6.2 Two-Dimensional Noise



0.6.2 Two-Dimensional Noise

```
void setup() {
  size(300,300);
}
void draw() {
  loadPixels();
  for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
      float bright = random(255);
//float bright = map(noise(x, y), 0, 1, 0, 255);
      pixels[x+y*width] = color(bright);
    }
  }
  updatePixels();
}
```





0.6.2 Two-Dimensional Noise

```
float xoff = 0.0;

void setup() {
  size(300,300);
}
void draw() {
  loadPixels();
  for (int x = 0; x < width; x++) {
        float yoff = 0.0;
     for (int y = 0; y < height; y++) {
        float bright = map(noise(xoff, yoff), 0, 1, 0, 255);
        pixels[x+y*width] = color(bright);
        yoff += 0.01;
    }
    xoff += 0.01;
  }
  updatePixels();
}
```