Engineering Analysis

Lec.4

Fall course 2021-2022

3rd Year

Nyan Dawood

Randomness and probability

Objectives :

- Review a programming concept central by OOP.
- The random walker will serve as a template for how to moving objects around a Processing window.

Random Walks

- An **object** in Processing is an entity that has both data and functionality.
- A **class** is the template for building actual instances of objects.

Let's begin by defining the Walker class, what it means to be a Walker object. Have x , y location.

```
class Walker {   int x;
                 int y;
```

Also must contains constructor .

- a **constructor** special function that is called when the object is first created.

```
  Walker() {
        x = width/2;                    // initialize first location in the
        y = height/2;                    // center of window

           }
```

- Walker class has two functions:

  first function that allows the object to display itself (as a white or black dot).

```
void render() {                    //function to display dot
          stroke(0);
          point(x,y);
                    }
```

Second function directs the Walker object to take a step.

There are four possible steps. X++, X--, Y++, Y–

,By randomly pick from four choices using **random().**

```
void step() {
          int choice = int(random(4));
if (choice == 0) {                                    //The random "choice" determines our step.
                  x++;}
else if (choice == 1) {
                  x--;}
 else if (choice == 2) {
                  y++;}
 else {
                  y--;}
}
}
```

in the main part of our sketch declare one global variable of type Walker.

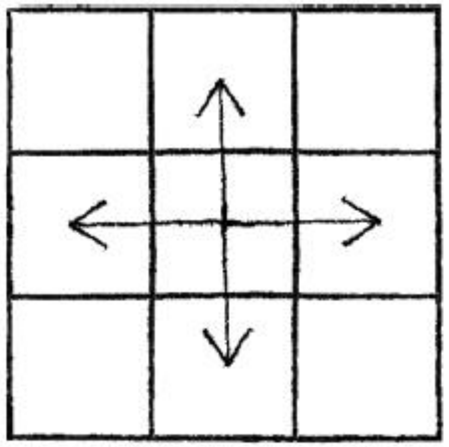
```
Walker w;
```

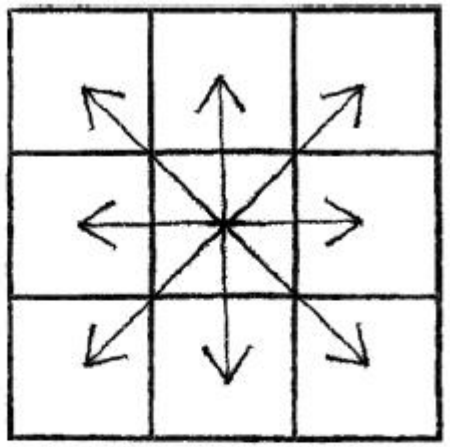
Then create the new object and setup window size and color

```
        void setup() {
         size(640,360);
         w = new Walker(); Create the Walker.
         background(255);
         }
```

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

This Walker’s step choices are limited to four options—up, down, left, and right. But any given pixel in the window has eight possible neighbors, and a ninth possibility is to stay in the same place.


4 possible steps


8 possible steps

Randomly step to eight possible neighbors

```
void step() {
int stepx = int(random(3))-1;            //Yields -1, 0, or 1
int stepy = int(random(3))-1;
x += stepx;
y += stepy;
}
```

Probability of four neighbor step = $\frac{1}{4}$ = 25% chance

Probability of eight neighbor or remaining in its location = $\frac{1}{9}$ = 11% chance

Probability and Non-Uniform Distributions

With a few tricks, we can change the way to use random() to produce "non-uniform" distributions of random numbers.by many ways:

First: fill an array with a selection of numbers—some of which are

repeated then choose random numbers from that array and generate events based on those choices.

```
int[] stuff = new int[5];
stuff[0] = 1;                         //1 is stored in the array twice
stuff[1] = 1;                        // probability to pick 1 will be 40%
stuff[2] = 2;                        // probability to pick 2 will be 20%
stuff[3] = 3;                        // probability to pick 3 will be 40%
stuff[4] =3;
int index = int(random(stuff.length));      //Picking a random element from an array
```

Second : allow an event to occur only if our random number is within a certain range.

Example: Let’s say that Outcome A has a 60% chance of happening, Outcome B has 10% chance, and Outcome C, a 30% chance.

- 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

Sol.

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

We could use the above methodology to create a random walker that tends to move to the right. Here is an example of a Walker with the following 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);
if (r < 0.4) {                    //40% chance of moving to the right!
x++;
} else if (r < 0.6) {
x--;
} else if (r < 0.8) {
y++;
} else {
y--;}
}
```

Exercise

Create a random walker with dynamic probabilities. As instance you can give it a 50% chance of moving in the direction of the mouse?