Project 1 – Random Walk

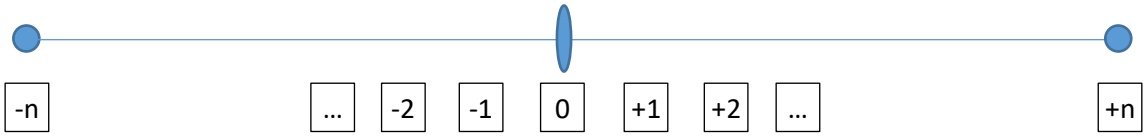
Overview

For this project, you are going to implement variations on what is called a Random Walk. A random walk is a process that represents some number of successive **steps** within some mathematical space. In this particular case you will work within 2-dimensional, Cartesian space, but the concept is applicable elsewhere as well. Random walks can be used in a variety of contexts, including studying probabilities of various sequences of events, but for this assignment we are going to focus on the visual results of this process.

For additional information on the concept: https://en.wikipedia.org/wiki/Random_walk

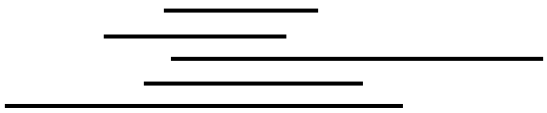
Simple Example – 1D Random Walk

In one-dimensional space, the process would be as simple as this: flip a coin—if it's heads, take a step to the left, and if it's tails, take a step to the right. After you've moved, repeat the process.



The maximum possible location after **n** steps would be either +n or –n, but the result is far more likely to be some somewhere in the middle, as you're unlikely to move only in one direction every time. (For example, flipping heads 100 times in a row, while not impossible, has a rather low probability of actually happening.)

If you were to plot the results of this one-dimensional random walk, it would simply be a line of varying length (assuming the same number of steps, and a sufficiently random method of generating numbers):



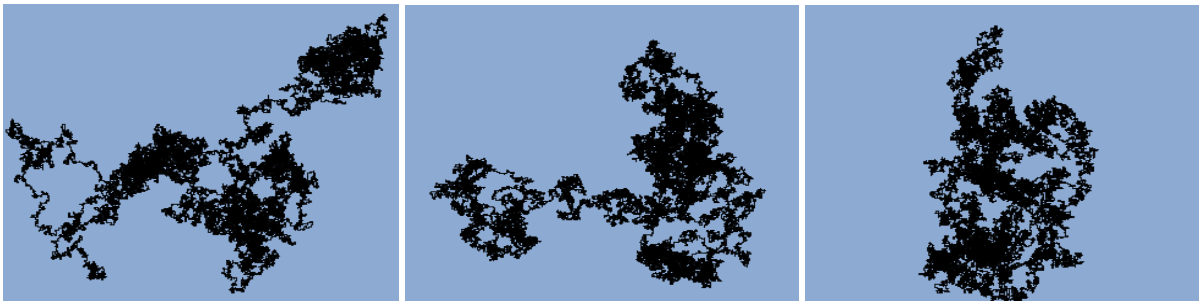
Not the most interesting results, but you've got to start somewhere

The reason the lines vary in length is because you will backtrack occasionally, stepping to locations you've already visited—unless your algorithm implements some method of avoid this. Revisiting locations isn't inherently good or bad; it depends on what your goal is.

2D Random Walk

With the addition of a second dimension, the process can get more interesting. The idea is still the same: generate a random number (no longer a coin flip, but a random value between 1 and 4), and

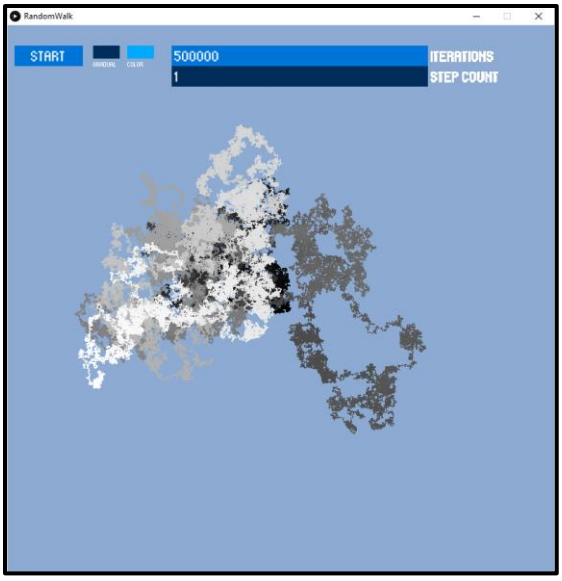
based on that number, move in a particular direction. For example, a 1 might be used to represent up, or north, or a decrease in the y-value of a point, while a 2 might represent right, east, an increase in the x-value of a point, and so on.



2D Random walks, with approximately 70,000 iterations each

Description

For this assignment you are going to create a program that looks something like the following:



The key features of this program will be:The UI, which consists of

- A Start button, to clear the previous result and start a new random walk
- 2 Toggles:
  - One to control whether or not the color ramps up based on the number of iterations
  - One to control whether the random walk is executed all at once, or gradually, one frame at a time
  - Terminology Note: A toggle is essentially a Checkbox
- 2 Sliders:
  - One to control the maximum number of iterations, with a range of 1000 to 500,000
  - One to control the number of steps to be executed at once in the gradual process, with a range of 1 to 1,000
- The rendered data itself, the visual result of the random walk algorithm you implement

Random Walk Algorithm - Moving

As mentioned earlier, the simplest form of the random walk algorithm is in one-dimensional space, and involves a "coin toss"—generating a random number that is either 0 or 1. Based on the result, move either left or right, and repeat as many times as you like, or as specified by some variable.

In 2-dimensions, the same concept applies. You can generate a random number between 1 and 4 (or 0-3, if you want to count like a programmer!) and then move accordingly:

```
// Take a step
Generate a random number from 0-3
if number is 0
   move up
else if number is 1
   move down
else if number is 2
   move left
else if number is 3
   move right

/* After you've taken a step, the next thing would be to draw the step. In this
assignment, that will be a simple point, using the point(x, y) function. */
```

You could leave it at that, but this runs the possibility of going out of bounds of your screen. That isn't inherently problematic, and in your own projects you may choose to treat that particular issue as not an issue at all (especially if you were running this with a large number of iterations, or tracking each point regardless of whether or not landed in some particular boundary).

For this assignment, however, you should **clamp** the values to ensure they always stay within a particular range. The range in this case would be 0 and the **width** for the x value, and 0 and **height** for the y value.

Random Walk Algorithm - Drawing

After you've taken a step, the next thing would be to **draw** the step. In this assignment, that will be a simple point, using the **point(x, y)** function. The color of the point is determined by the last value(s) passed to the **stroke()** function.

UI Specifications

Structure your UI so that it matches the image earlier in this document. Small discrepancies such as the position of a control, text size, etc are acceptable, but try to match the layout as closely as possible.

Window Size: Create a window of size(800, 800)

Start Button

The start button should do just that: start the execution of your algorithm. Clear the screen of any previous results, reset any necessary values, and begin. Whether your program is running gradually or all at once, it should begin with the press of this button.

Specifications: The random walk should always start in the center of the screen (the X and Y values should be half the screen width and height, respectively). Previous results should be cleared.

Toggle - Color

The first of two Toggle objects determines whether or not the steps are colored, based on their current iteration. The process you can use for this centers around a very useful function called **map()**. This function takes a value and converts it from one range to another, which you will often have to do when data translating data to some other context. https://processing.org/reference/map_.html

For example, colors are typically represented as either integer values with a range of 0-255 OR as floating-point values 0-1.0 for red, green, and blue components. You could write such a conversion function yourself, but the **map**() function already exists to do that work for you. Always utilize the tools you have available in a particular API!

You can use this function to generate a color value (ranging from 0-255) based on the number of iterations, or steps, the random walk has already generated. So the earliest steps will have a color value of black (or close to it), and the last steps will have a color value of white (or close to it).

****SHORTCUT**** The various color-related functions like stroke() and fill() can be used with a single parameter to create grayscale colors, or with three parameters to control the R, G, B values individually.

Specifications: The first step should be black (0, 0, 0), and all subsequent steps increasing in color value until the final step is white (255, 255, 255).

Toggle – Gradual

The second of the two Toggle objects controls whether or not the execution of the random walk will happen all at once in a single loop, or gradually, one frame at a time. How you choose to implement that behind the scenes is up to you.

Slider – Iterations

This slider determines the total number of iterations, or steps, your random walk will perform. The minimum number of iterations should run fairly quickly even on less-powerful hardware, while the maximum number might take a moment or two.

Specifications: The range of the slider should be from 1000 to 500,000.

Slider – Step Count

This slider sets a counter to indicate how many steps should be performed in a single frame, if the execution mode is set to run gradually. Verifying 20,000 steps running successfully, one at a time, would be a time-consuming process. This gives you control over how much you can speed that process up.

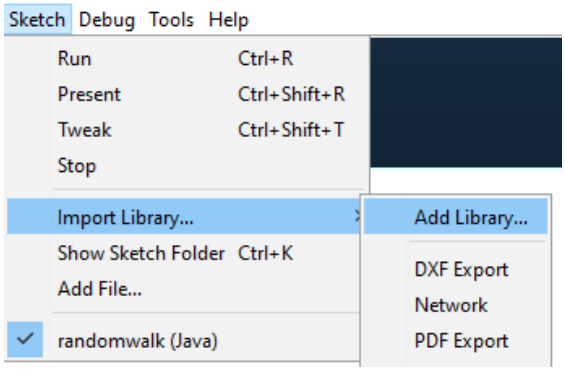
Specifications: The range of the slider should be from 1 to 1,000.

ControlP5

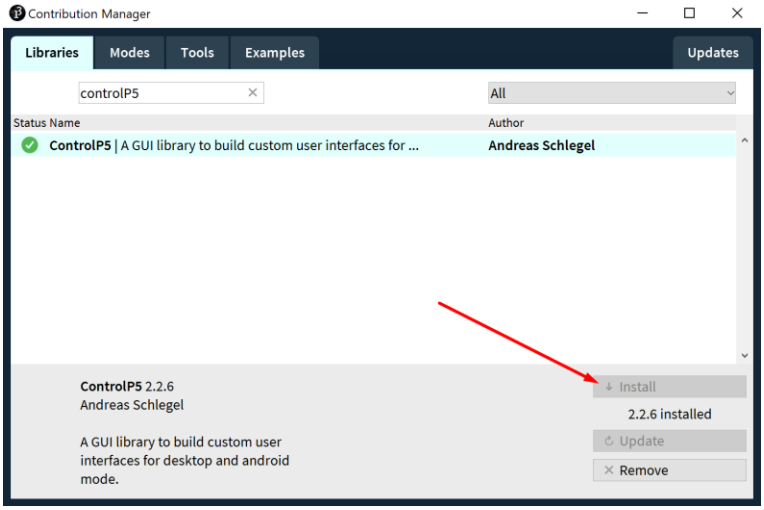
Processing allows you to create custom libraries which you (or others) can then import as needed. There are a variety of libraries out there, but you'll use one in particular for this project: **ControlP5**. ControlP5 is a user-interface (UI) library. Pretty much every application needs a UI of some sort, with all the various controls that come along with that—buttons, checkboxes, text fields, etc. As described in the previous section, you will only need a small handful of controls for this project.

To install ControlP5 is a simple process:

First, you need to import the library. Click on **Sketch->Import Library…->Add Library…**



In the next screen, type ControlP5 in the Filter box. Click ControlP5 in the list, and the click the Install button.



Once that's finished, you're almost done! The library is now installed, and in order to use it in your projects you have to have an **import** statement in your code files (typically right at the top of the file). The line for ControlP5 is:

```
import controlP5.*;
```

That's all there is to it! Now that you've imported the library, you can actually get to work utilizing it. You can find extensive examples on how to use the various controls in your project here: http://www.sojamo.de/libraries/controlP5/#examples

Look at a few of the examples, and you will see that they follow a similar pattern. First, they require a ControlP5 object to be created; this is the primary connection between your application and the UI library. The constructor of this class takes **this** (your main sketch class where all of your code resides by default) as its only parameter.

Individual controls are then added to this particular object, and the functions to add those controls return a reference to the control that was just created. For example, if you wanted to create a Button,

the function addButton(…) would return a reference to that newly created Button control. If you catch that, you can then access the numerous functions to modify that button—size, position, text, etc.

All of the information you need to use this interface library is there on the examples page, as well as the JavaDoc reference page. Learning to navigate documentation to find examples, or the answers you seek for a particular problem, is absolutely a skill you must develop, if you haven't already. This cannot be stated enough! Every external tool or library you work with will have its own unique functionality, which you will have to learn in order to properly use it.

Submissions

Create a .zip file with any code files you created for this project (in Processing they are files with the extension .pde), and name the file ***LastName.FirstName.Project1.zip***. Submit the .zip file on the Canvas page for Project 1.

Tips

- Use the reference page! Processing has a very thorough reference at https://processing.org/reference/
- Ditto for ControlP5. The reference and example pages are very thorough. Using a new tool or library can initially be a bit overwhelming, but step by step you'll learn more about it.
- Points are colored by the stroke() function, not the fill() function. Fill() is used for shapes (i.e. something with area).
- Tackle one thing at a time. Even in a small project such as this, breaking it into smaller pieces can help you to make sense of the larger application.

Grading

Item	Description	Maximum Points
UI	Start Button, 2 Checkboxes, 2 Sliders all functional	20
Random Walk - Immediate	Random Walk completes all at once, a total number of steps as indicated by the iterations slider	20
Random Walk – Gradual	Random Walk completes gradually, one frame a time. The number of steps taken each frame is determined by the step count slider	40
Color	Steps are colored according to the number of iterations	20
	Total	100