▼

The Craft of Pseudocode

Creating with Code

Prototype Development

ideas **+** background **+** understanding of medium **= concept development**

concept development **+** skills **+** experimentation **= prototype development**

DATA & CODE

The digital is inherently flexible & mutable. By understanding code, and learning how to read, manipulate and display data you have true creative play in the digital environment.

DIGITAL RUBE GOLDBERG PROCESSOR

"Given the fact that any stored data is binary code in the end, ***IT IS THE ENCODING AND DECODING ALGORITHMS THAT MAKE DIGITAL DATA MEANINGFUL FOR US***. To create and awareness for that, we came up with the idea of the rube-goldberg-processor. It is an potentially endless line of sub-processor that transform the same dataset from one start to another"

- Product Collective (http://www.the-product.org/)

ALGORITHM

Not such a scary word after all.

‘An algorithm is a procedure or formula for solving a problem. In computer programming an algorithm is the sequence of steps required to perform a task”

- Learning Processing, Shiffman

MAKE TEA ALGORITHM

1. Get Cup
2. Put teabag in cup
3. Boil Water
4. Pour water into cup
5. Brew
6. Add milk
7. Remove teabag
8. Add sugar (how many)
9. Stir

Algorithm

1. Cut a a four-sided polygon from the sheet.

2. Attach it to an existing polygon so that a pair of three polygons forms together one edge that is as straight as possible.

RULE: A branch must not contain more than one polygon of each color.

3. go to 1.

PSUEDOCODE

When you write pseudocode, you are defining the algorithm of your idea. It involves writing/drawing/playing out the structure of your idea step by step.

SOFTWARE STRUCTURES

Wall Drawing #797

The first drafter has a black marker and makes an irregular horizontal line near the top of the wall. Then the second drafter tries to copy it (without touching it) using a red marker. The third drafter does the same, using a yellow marker. The fourth drafter does the same using a blue marker. Then the second drafter followed by the third and fourth copies the last line drawn until the bottom of the wall is reached.

- by Sol Le Witt

LOW & HIGH RESOLUTION

Low:
Mapping the flow of the program in words. What does it do over time. Mapping the interaction if it is interactive. Work out what the participant does, and what happens in response.

High:
Breaking down the flow/behavior into smaller parts and then breaking this down even further into steps in pseudo code. Then translating into real code. Working out what parts you don’t know how to do. Find relevant examples for these.

LOW RESOLUTION

The Input:
What does your program respond to? In the case of your assignment it is the progress of time. Does your program respond to a participant?

The Response:
What happens in response? What changes as seconds, minutes and hours change? What happens (if anything) when the participant responds?

LOW RESOLUTION

• Test and experiment how your concept or ideas would work (or don't work)

• Think iteratively. Your idea is still fluid, it can change in response to your discoveries in low resolution.

- Communicates your idea clearly
- Identifies what you need to map

SOFTWARE STRUCTURES

Software Structure #003

A surface filled with one hundred medium to small sized circles. Each circle has a different size and direction, but moves at the same slow rate. Display:
A. The instantaneous intersections of the circles
B. The aggregate intersections of the circles
- Casey Reas

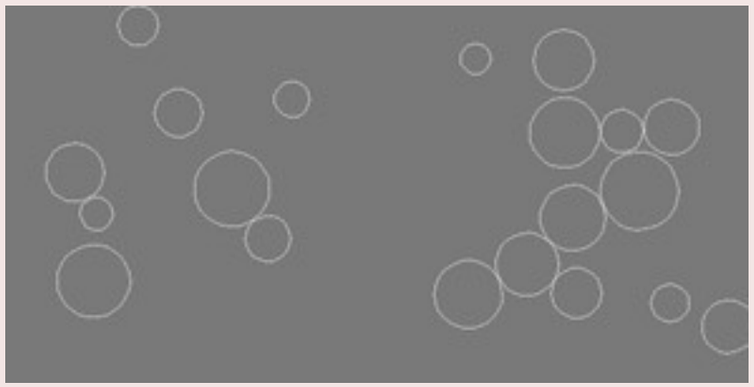
(http://artport.whitney.org/commissions/softwarestructures/map.html)

PROCESS

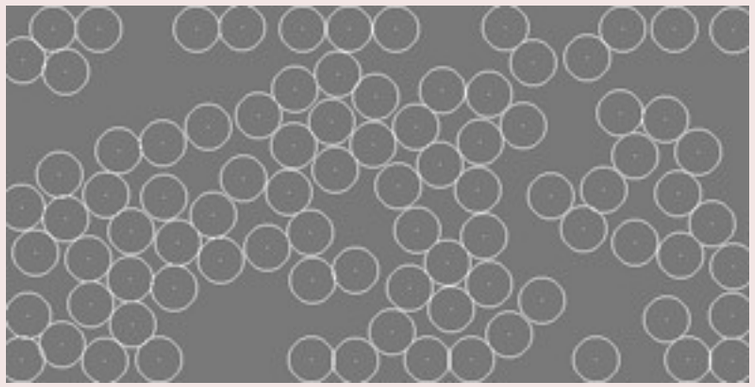
1. Idea
2. Algorithm low resolution, sketch or enact
3. Parts - Break the idea down into smaller paers
    a. Algorithm Pseudo code - for each part work out the algorithm for that part in pseudocode
    b. Algorithm Code - Implement the pseudocode in real code
        i. Modify
        ii. Iterate
    c. Objects - make the algorithm into a class (we haven't covered this yet)
4. Integrate - take all the parts and get them working in one algorithm

PARTS

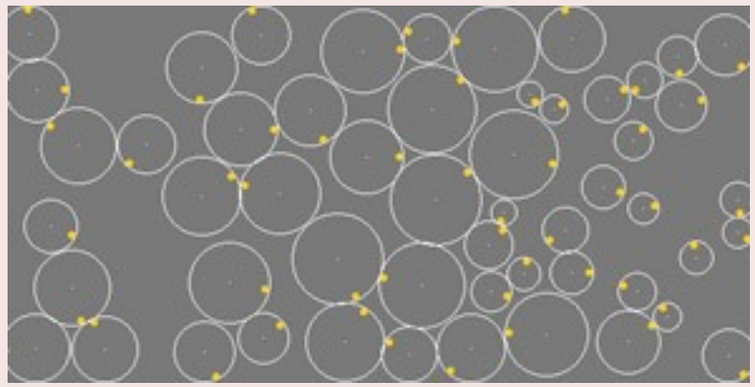
1.



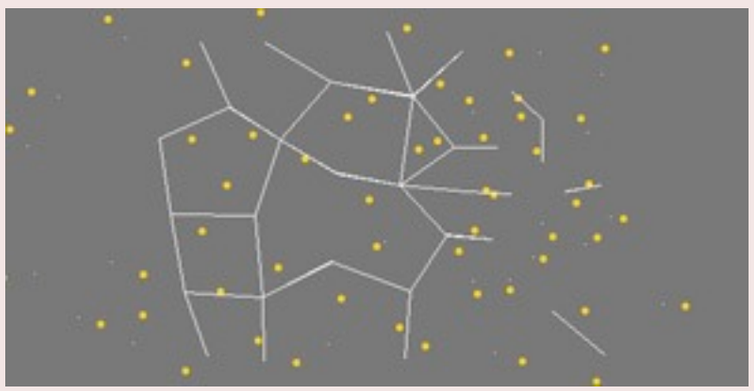
2.



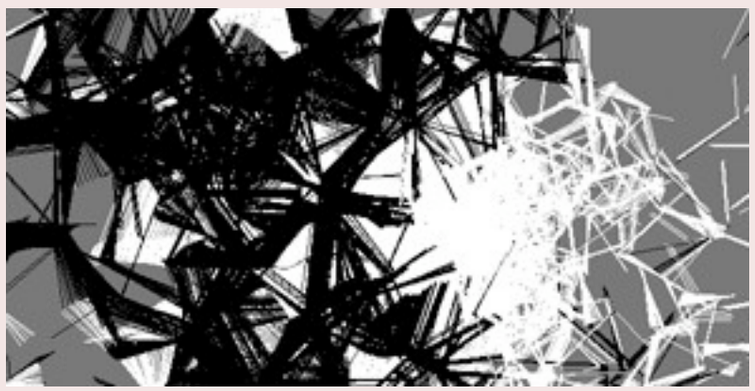
3.



4.



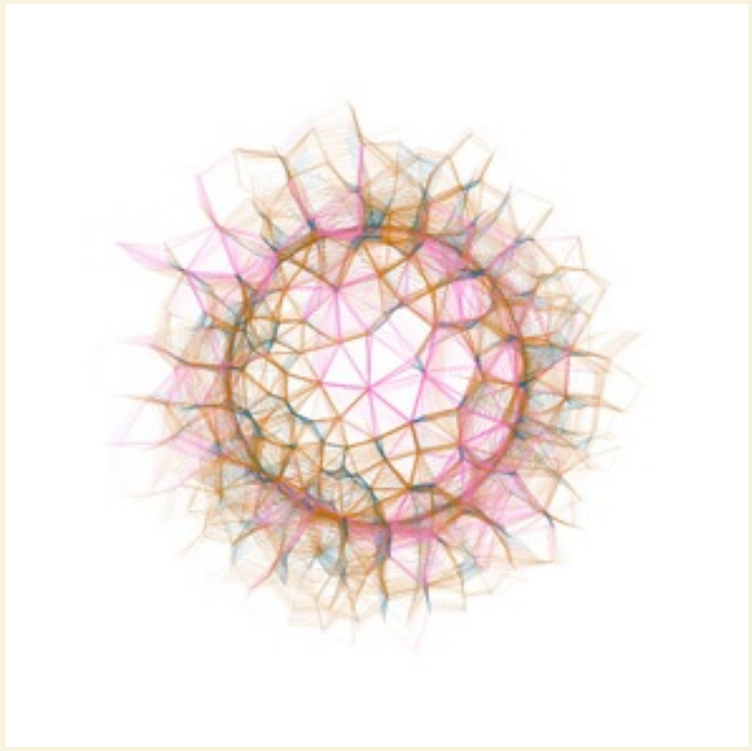
5.

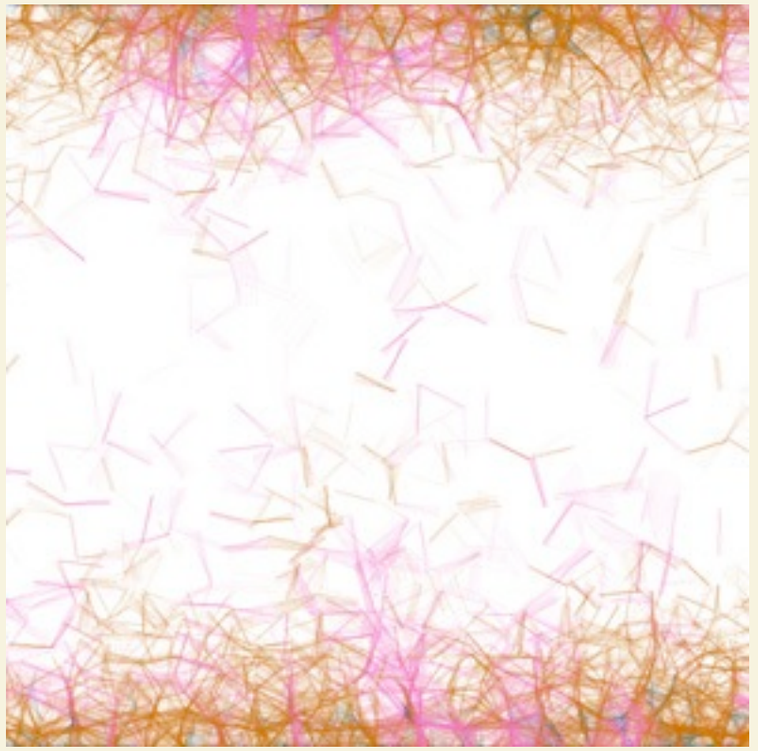


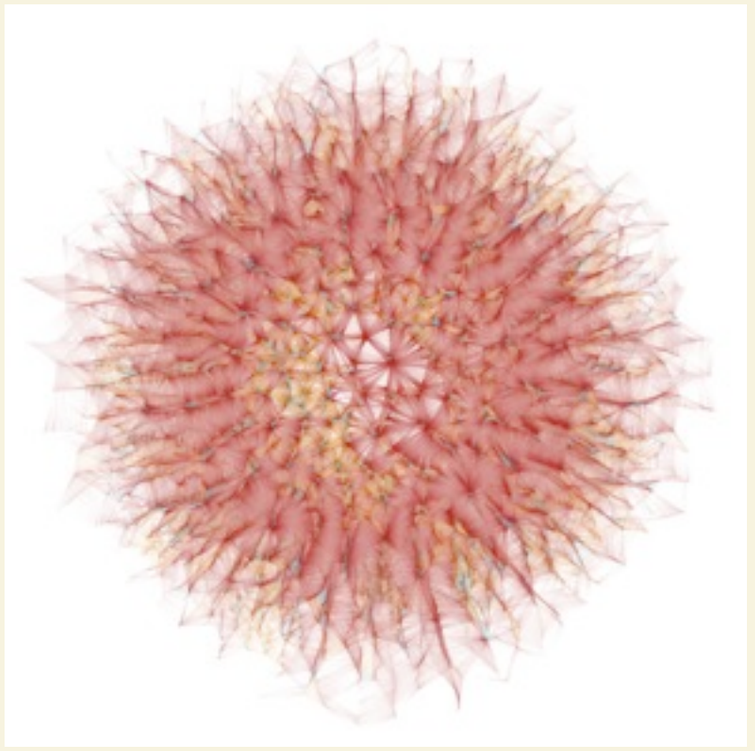
http://artport.whitney.org/commissions/softwarestructures/text.html#process

http://artport.whitney.org/commissions/softwarestructures/map.html

ITERATION







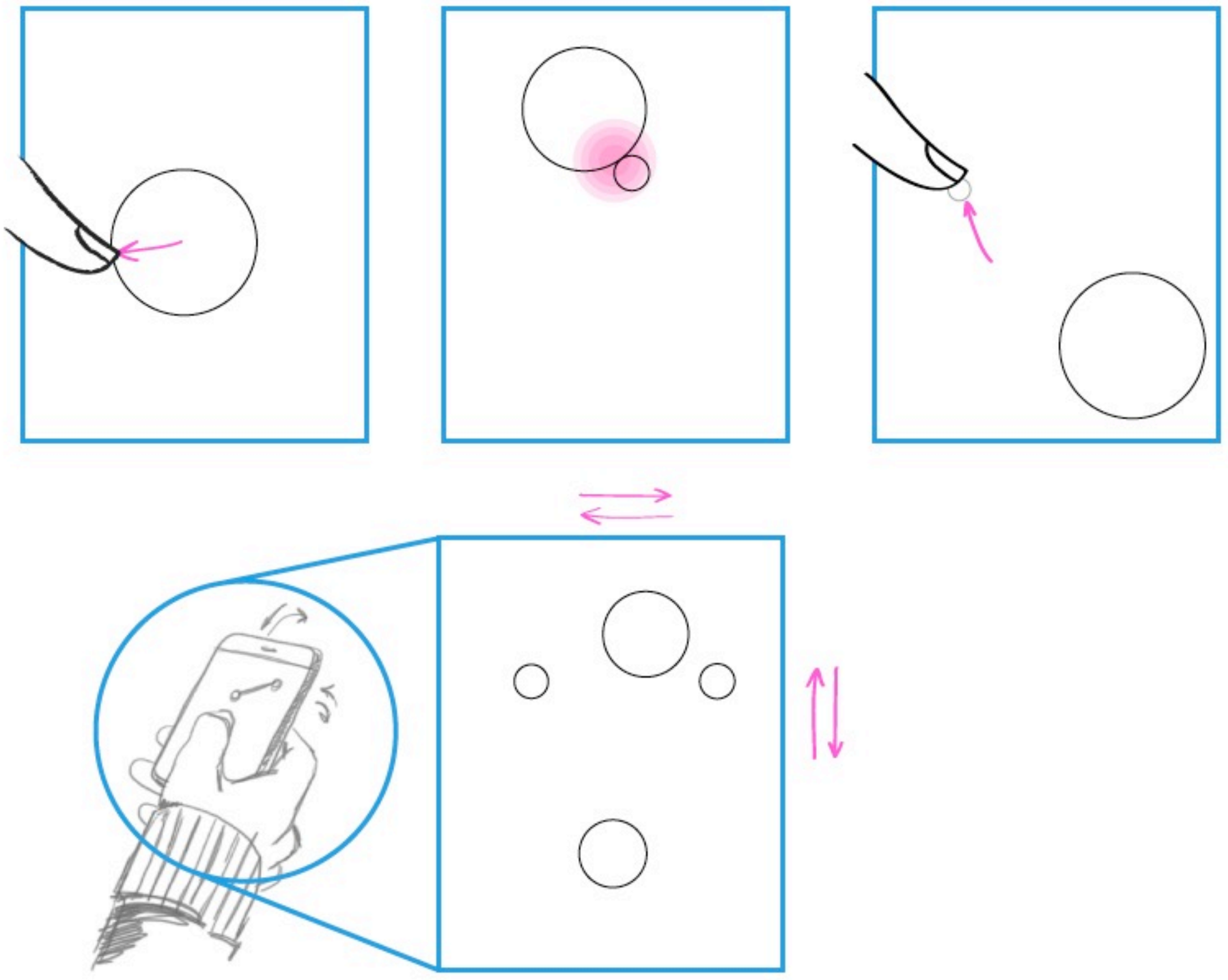
Visual iterations from the one “software structure”

CONCEPT

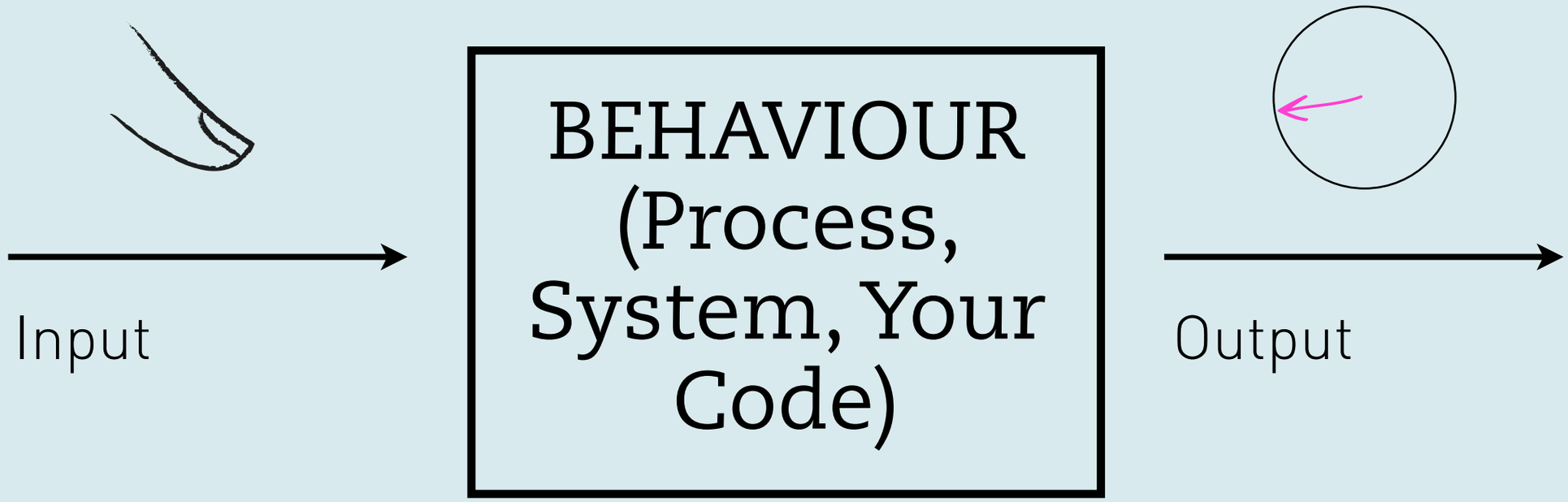
Timbre

Timbre is what makes one sound different from another, even when they have the same pitch and volume. i.e. the difference between a guitar and a violin. From French, “quality of sound”, and earlier “sound of a bell”

Users can draw a circles of different sizes onto the screen, each circle is given a sound. The circles float around the screen, this can be affected by tilting the device. As circles hit each other they play their sound relative to the other circle. i.e. if one is long and one short, one will be quiet and the other loud. If they are the same size, equally loud. Pitch is determined by speed. Size can be edited by dragging within the circle. Circles can be removed by dragging them to a small size.



BEHAVIOUR



WHAT WE NEED TO WORK OUT

Part 1. Write a program where the user can draw a circle by clicking and dragging

Part 2. Write a program that where the user can press on a circle and drag it to a new width. If they drag it below N width, it disappears

Part 3. Write a program to test if two circles intersect (eventually bounce/sounds will play in response to this)

Part 4. Write a program where circle[s] moves around the screen.

part 1: Draw a Circle

1. Erase background
2. When the user presses down on the mouse, record the location (this will be the centre of the circle)
3. As the user drags. Record the distance from the centre location to the mouse .
4. Draw a circle from the centre location. Its radius will be the distance from the centre to the mouse.

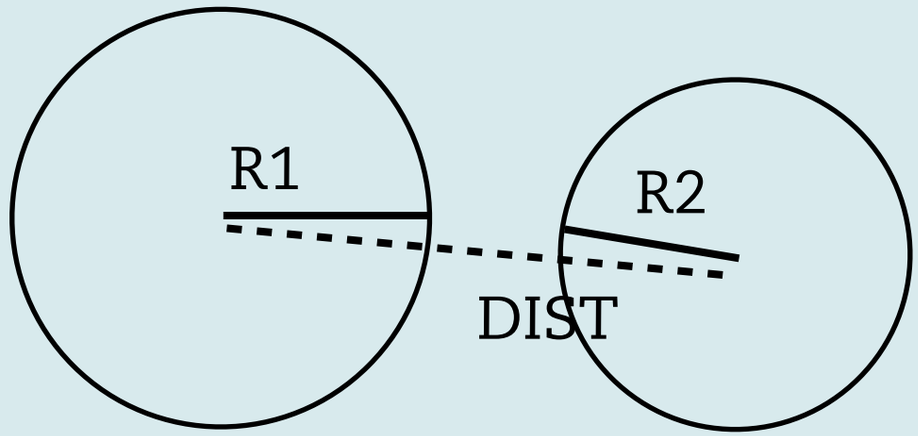
part ii: Change Circle Size

1. Erase background
2. Draw a circle in the centre of the screen at R radius
3. If the user clicks & drags within the circle, set R radius to the distance from the centre of the circle to the mouse location.
4. When the user releases the mouse, if the R is less than 10, remove the circle.

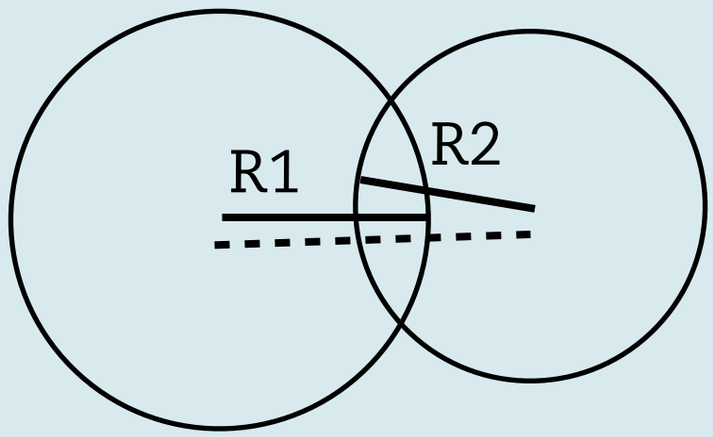
part iii: Intersecting Circles

1. Erase background
2. Draw two circles
3. One circle is stationary, at X an Y location with R1 radius
4. The other circle moves with the mouse and has R2 radius
5. If circle 1 intersects with circle 2, turn both white.
   Otherwise they will be black

part iii: Intersecting Circles



If the dist is greater than r1 + r2 the circles are not intersecting



If the dist is less than r1 + r2 the circles are not intersecting

part iv: Moving Circles

a. A single moving circle, when it hits the edge of the display window it moves in the opposite direction
b. A single moving circle which is affected by the mouse position
c. An array of moving circles
d. An array of moving circles that are affected by the mouse position
e. Fancier moving circles (iteration)