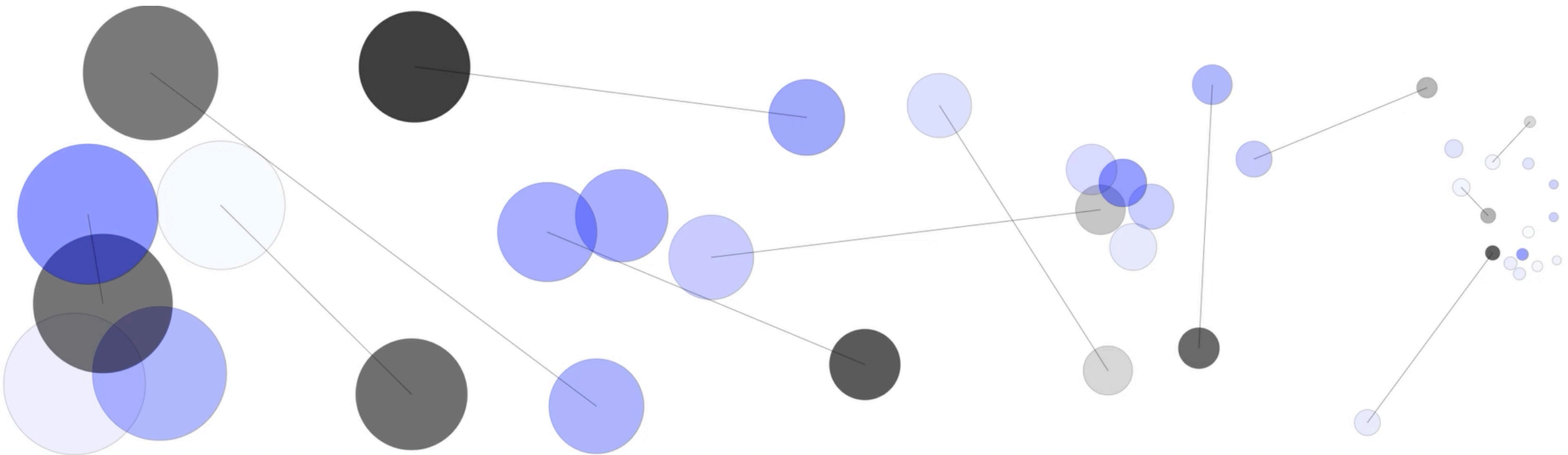


an introduction to the WEB AUDIO API

Day 4

highSCORE



Alberto Barberis
2021

highSCORE

7. JAVASCRIPT DEALING WITH INTERACTION - 2

Dot Drone Generator is a drone generator which allows you to create synthetic textures and chords directly on your browser.

Click on the window to generate a Dot, which has a sinusoidal wave with tremolo.

The y-axis represents the amplitude range. The amplitude is modulated by a triangular LFO (Low Frequency Oscillator), with a random frequency.

The x-axis represents the frequency range.

Press 'L' and then Click+Drag from an existing Dot to an other one, to link two sinusoids and create a Frequency Modulation (FM synthesis) between them, where the first Dot becomes (in addition) the modulator of the second (carrier).

It is possible to create a chain of modulation: each carrier can become a modulator. This allows you to create complex spectra with a lot of sidebands, to the point of creating very noisy sounds!

Let's CODE!!

```
/** ////////////////////////////////////// */
 * SUPPORT FUNCTIONS
 * definition
 * ////////////////////////////////////// */

function startAudio(){ // audio to start when the first dot is created
    audioContext = new AudioContext(); // create an audiocontext
    masterGain = audioContext.createGain(); // create a gain Node for the master gain
    masterGain.gain.value = MASTER_GAIN; // set the level of the master gain (gain adaptation)
    masterGain.connect(audioContext.destination); // connect the master gain to the destination
}

function deleteModulatorCarrier(){
    modulator = null; // remove the object from the variable modulator
    carrier = null; // remove the object from the variable carrier
}
```

We create a new function `deleteModulatorCarrier()` that we use to empty the modulator and the carrier variables.

The goal is to store in these variables always a different pair of modulator/carrier.

The modulator will be the Dot Object that is selected first when the user type click+Drag+L. The carrier is the second Dot Object over which the mouse is passed while click+Drag+L.

```
function deleteModulatorCarrier(){
  modulator = null; // remove the object from the variable modulator
  carrier = null; // remove the object from the variable carrier
}

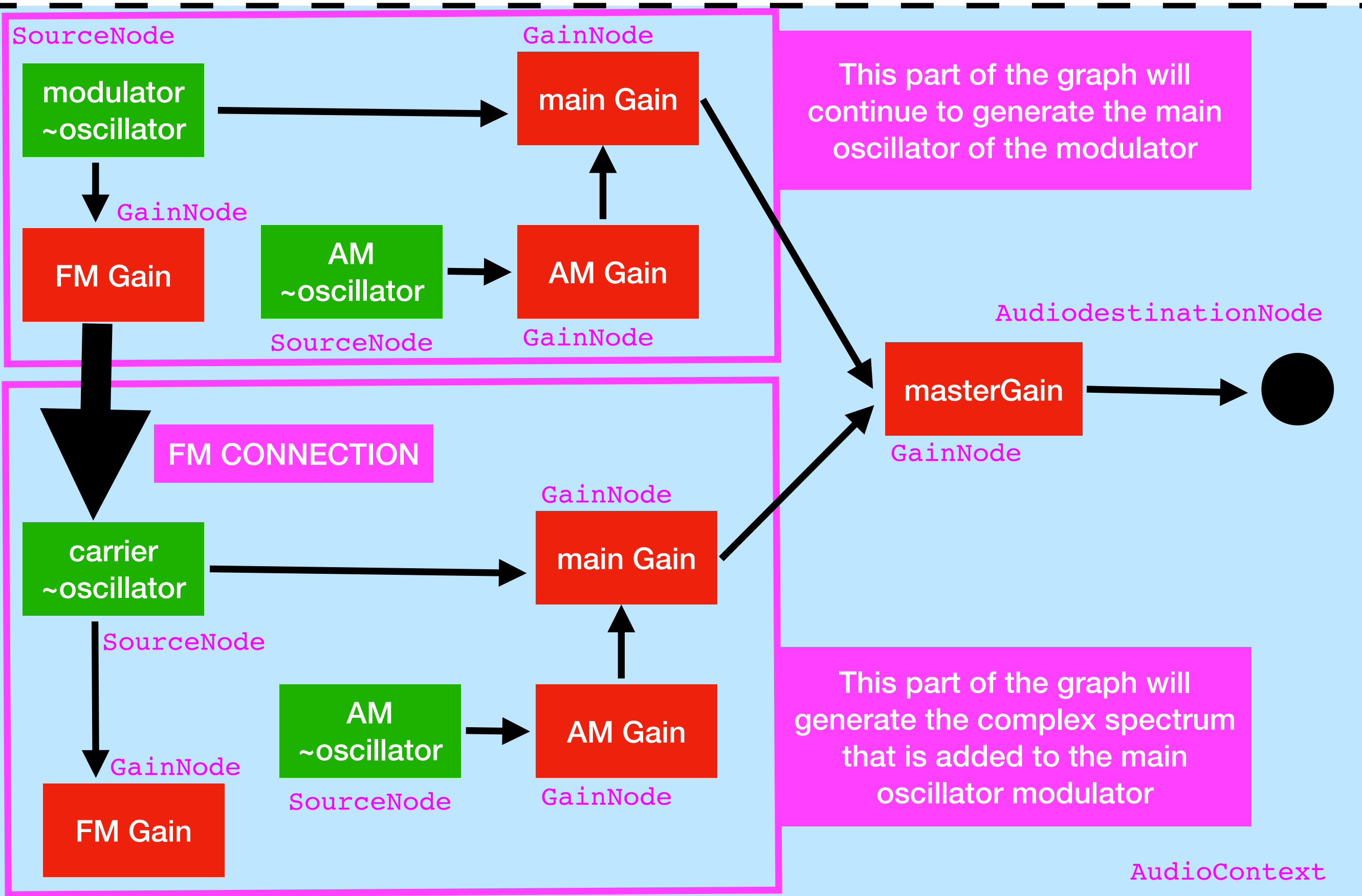
function applyFM(modulator, carrier){ // function that activate the FM synthesis
  modulator.FMoscGain.connect(carrier.mainOsc.frequency); // connect the FMoscGain to the frequency of the carrier main Oscillator (this is the FM)
  modulator.FMoscGain.gain.setTargetAtTime(MODULATION_INDEX, audioContext.currentTime , ATTACK_TIME); // set the FMoscGain to the modulation index value
}
```

Let's CODE!!

We create the function `applyFM()` that we use to start the FM between the modulator and the carrier. The function has two parameters: the modulator and the carrier.

To implement the FM we have to connect the FM oscillator Gain to the carrier main oscillator frequency.

After that we have to start the FM oscillator Gain (that is set to zero initially) setting it to the `MODULATION_INDEX` constant (the value that controls the amount of sidebands and their amplitudes).



Let's CODE!!

```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
  if(!keyIsDown(76)) return; // exit from the function if is not pressed the key 'L'
```

Here we implement an other function of the p5.js library: the `mouseDragged()` function is called every time the user click + drag the mouse.

In the first line of this function we use an other function of the p5.js the `keyIsDown()`. This function returns true if the user type a certain letter in the keyboard; the letter is passed as `keyCode` parameter (we use the “L” for “Link” that has 76 as `keyCode`).

If the keyboard letter is not the “L” we exit from the function. The “!” means NOT.

Let's CODE!!

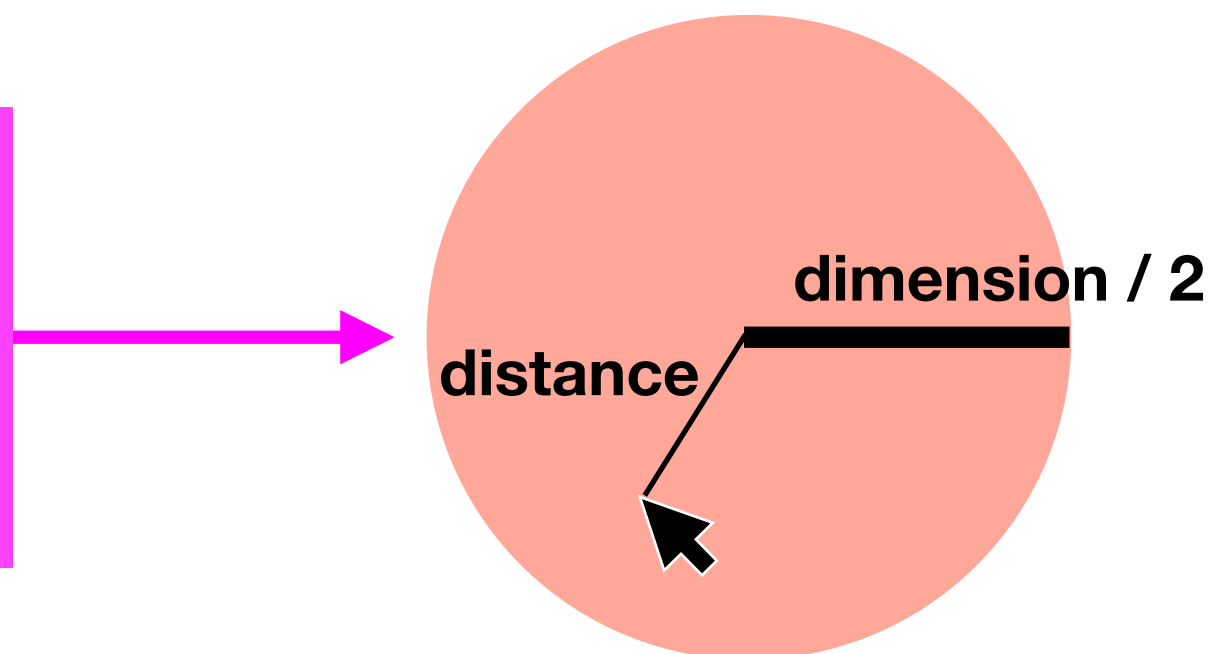
```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
  if(!keyIsDown(76)) return; // exit from the function if is not pressed the key 'L'

  for(let i=0; i<arrayOfDots.length; i++){
    let currentDot = arrayOfDots[i];
    let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the current Dot center and the mouse
```

Here we create a for loop that for each Dot calculate the distance between the current x and y positions of the mouse from the x and y positions of the Dot center.

If the distance is less than the dimension/2 of the Dot (i.e. the radius of the circle) we can say that the mouse position is inside a certain Dot.

We can say that the mouse is **INSIDE** the Dot because the distance between the mouse and the center is less than the radius (dimension / 2)



7.1 BOOLEAN EXPRESSIONS

A **Boolean expression** is an expression used in programming that produces a Boolean value when evaluated.

A Boolean value is either **true** (1) or **false** (0).

The most common **Boolean operator** are:

- NOT (!)
- OR (||)
- AND (&&)

The Boolean algebra follows the **Truth Table**.


A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Let's CODE!!

```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
  if(!keyIsDown(76)) return; // exit from the function if is not pressed the key 'L'

  for(let i=0; i<arrayOfDots.length; i++){
    let currentDot = arrayOfDots[i];
    let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the current Dot center and the mouse

    if(!modulator && !carrier && d<currentDot.dimension/2 && !currentDot.connections[1] ) {
      modulator = currentDot;
      break; // exit from the for loop
    }
  }
}
```



Here we check the following conditions with one boolean expression:

- the modulator does not exist (null is false in javascript)
- the carrier does not exist
- the distance is less than the radius
- and the current Dot does not already have a carrier (we fetch the second Object in the Array connections)

If all the conditions are verified then: we found a modulator! And we store it in the right variable. We exit also from the current loop cycle, because if the Dot is a modulator it can not be a carrier and it does not make sense to proceed.

Let's CODE!!

```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
  if(!keyIsDown(76)) return; // exit from the function if is not pressed the key 'L'

  for(let i=0; i<arrayOfDots.length; i++){
    let currentDot = arrayOfDots[i];
    let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the current Dot center and the mouse

    if(!modulator && !carrier && d<currentDot.dimension/2 && !currentDot.connections[1] ) {
      modulator = currentDot;
      break; // exit from the for loop
    }

    else if(modulator && !carrier && d<currentDot.dimension/2 && !currentDot.connections[0] && arrayOfDots.indexOf(modulator)!=i){
      carrier = currentDot;
      modulator.connections[1] = carrier;
      carrier.connections[0] = modulator;
      applyFM(modulator, carrier);
      break; // exit from the for loop
    }
  }
}
```

Here we check the following conditions with one boolean expression:

- the modulator exists
- the carrier does not exist
- the distance is less than the radius
- and the current Dot does not already have a modulator (we fetch the first Object in the Array connections)
- and the index of the already found modulator in the arrayOfDots is not the index of the current iteration (otherwise we would find that the modulator is also the carrier)

Let's CODE!!

```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
  if(!keyIsDown(76)) return; // exit from the function if is not pressed the key 'L'

  for(let i=0; i<arrayOfDots.length; i++){
    let currentDot = arrayOfDots[i];
    let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the current Dot center and the mouse

    if(!modulator && !carrier && d<currentDot.dimension/2 && !currentDot.connections[1] ) {
      modulator = currentDot;
      break; // exit from the for loop
    }

    else if(modulator && !carrier && d<currentDot.dimension/2 && !currentDot.connections[0] && arrayOfDots.indexOf(modulator)!=i){
      carrier = currentDot;
      modulator.connections[1] = carrier;
      carrier.connections[0] = modulator;
      applyFM(modulator, carrier);
      break; // exit from the for loop
    }
  }
}
```

If the conditions are all verified, it means that we found a carrier.

We store the Dot Object in the variable.

We say to the modulator what Dot Object is the carrier and viceversa.

Then we apply the FM calling the function we already defined and passing it, as parameters, the modulator and the carrier.

```
function mousePressed() { // functtion called if the mouse is pressed
  if(!audioContext){ startAudio(); } // if the audio context still does not exists create it
  if(mouseY < 0 || keyIsDown(76)) return; // exit if the mouse is out of the canvas OR the user is pressing "L"

  let newDot = new Dot(mouseX,mouseY); // create the Dot calling the constructor of the class
  arrayOfDots.push(newDot); // put the Dot in the array of Dots
}
```

Let's CODE!!

Now we go back to our `mousePressed()` definition. Because we want that if the user is pressing L, the `mousePressed()` would not create a new Dot: we just want the FM!

We create a boolean expression: if the user is pressing the mouse out of the canvas or is typing "L" we exit from the function.

```
// when the mouse is released or a key is released delete the stored modulator and carrier  
function mouseReleased() {  
  deleteModulatorCarrier();  
}  
function keyReleased(){  
  deleteModulatorCarrier();  
}
```

Let's CODE!!

Now we implement other two functions of p5.js. Both of these functions call the function `deleteModulatorCarrier()` that we already implemented.

In this way every time the user release the mouse (`mouseReleased()`) or the key "L" (`keyReleased()`), the process of finding modulator and carrier will stop, emptying the variables and making them available for a new search.

```
function draw(){ // this is the draw function of the p5.js called to draw on the canvas
  clear(); // clear the canvas
  fill('rgba(255,0,0,0.3)'); // the color of the Dot: red with alpha
  strokeWeight(2); // the weight of the border of the elements

  for(let i=0; i<arrayOfDots.length; i++){ // draw each Dot in the array
    let currentDot = arrayOfDots[i];

    ellipse(currentDot.x, currentDot.y, currentDot.dimension, currentDot.dimension); // draw a circle

    let carrier = currentDot.connections[1]; // the carrier of the current Dot
    if(carrier){ // if the carrier exists draw a line between the modulator (current Dot) and the carrier
      line(currentDot.x, currentDot.y, carrier.x, carrier.y); // draw the line
    }
  }
}
```

Let's CODE!!

Now we go back to the `draw()` function.

Here we check if the current Dot in the loop cycle has a carrier, that means that the Object inside `connections[1]` is not null.

If so, we want to draw a line from the current Dot center to the carrier center. We do this with the p5.js function `line()`.

Now we can start to create some FM paths in the 2D space of Dots.

Dot Drone Generator is a drone generator which allows you to create synthetic textures and chords directly on your browser.

Click on the window to generate a Dot, which has a sinusoidal wave with tremolo.

The y-axis represents the amplitude range. The amplitude is modulated by a triangular LFO (Low Frequency Oscillator), with a random frequency.

The x-axis represents the frequency range.

Press 'L' and then Click+Drag from an existing Dot to an other one, to link two sinusoids and create a Frequency Modulation (FM synthesis) between them, where the first Dot becomes (in addition) the modulator of the second (carrier).

It is possible to create a chain of modulation: each carrier can become a modulator. This allows you to create complex spectra with a lot of sidebands, to the point of creating very noisy sounds!

Click on an existing circle to delete it or to delete the modulation chain of which it is part.

Here we implement a function that delete a Dot and all the Dots possibly connected with it (FM links)

Let's CODE!!

```
function deleteDot(currentDot){ // this is a recursive function that delete a dot from the array
  let indexToDelete = arrayOfDots.indexOf(currentDot); // fetch the index of the Object

  currentDot.mainOscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the main osc Gain
  currentDot.AMoscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the AM osc Gain
```

This is a **RECURSIVE FUNCTION**, that means a function that can call itself. The method `indexOf()` returns the index of the Current Dot. We need this later to delete the element from the Array.

Then we set to 0 the main oscillator Gain and the AM oscillator Gain. This action stops the sound of the Dot.


```
function deleteDot(currentDot){ // this is a recursive function that delete a dot from the array
  let indexToDelete = arrayOfDots.indexOf(currentDot); // fetch the index of the Object
  if(indexToDelete == -1) return; // if the array does not contain the current Dot

  currentDot.mainOscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the main osc Gain
  currentDot.AMoscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the AM osc Gain

  let modulator = currentDot.connections[0]; // store in a variable the modulator associated to the Object
  let carrier = currentDot.connections[1]; // store in a variable the carrier associated to the Object

  arrayOfDots.splice(indexToDelete, 1); // remove the Object from the array
```

Let's CODE!!

We fetch the modulator and the carrier that are possibly connected to the current Dot.

The method `splice()` of the class `Array` of javascript can be used to delete an element from an Array.

The first parameter is the index of the Array from where delete and the second parameter specifies how many elements we want to delete. In our case we just want to delete one element, the current Dot itself.

```
splice(start, deleteCount)
```

Let's CODE!!

```
function deleteDot(currentDot){ // this is a recursive function that delete a dot from the array
  let indexToDelete = arrayOfDots.indexOf(currentDot); // fetch the index of the Object
  if(indexToDelete == -1) return; // if the array does not contain the current Dot

  currentDot.mainOscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the main osc Gain
  currentDot.AMoscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the AM osc Gain

  let modulator = currentDot.connections[0]; // st
  let carrier = currentDot.connections[1]; // stor

  arrayOfDots.splice(indexToDelete, 1); // remove

  if(modulator && !carrier){ // if only the modula
    modulator.connections[1] = null;
    deleteDot(modulator); // recursion of the sa
  } else if (modulator && carrier){ // if both the
    modulator.connections[1] = null;
    carrier.connections[0] = null;
    deleteDot(modulator);
    deleteDot(carrier);
  } else if (!modulator && carrier){ // if only th
    carrier.connections[0] = null;
    deleteDot(carrier);
  }
}
```

We want to delete also all the Dots connected to the current Dot. We do the following using the RECURSION:

- if the current Dot has only a modulator, we delete it after emptying the variable that contains the carrier;
- if the current Dot has both a modulator and a carrier, we empty the correct elements in the array connections and then we delete both the modulator and the carrier;
- if the current Dot has only a carrier, we delete it after emptying the variable that contains the modulator;

Let's CODE!!

```
function mousePressed() { // function called if the mouse is pressed
  if(!audioContext){ startAudio(); } // if the audio context still does not exists create it
  if(mouseY < 0 || keyIsDown(76)) return; // exit if the mouse is out of the canvas OR the user is pressing "L"

  for(let i=0; i < arrayOfDots.length; i++){ // check if the mouse is pressed in an existing Dot; if so delete it

    let currentDot = arrayOfDots[i];
    let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the mouse and the current Dot

    if(d < currentDot.dimension/2){ // if pressing on an existing Dot (and key is pressed)
      deleteDot(currentDot); // delete the Dot
      return;
    }

    let newDot = new Dot(mouseX,mouseY); // create the Dot calling the constructor
    arrayOfDots.push(newDot); // put the Dot in the array of Dots
  }
}
```

Every time the user press the mouse, we check with a for loop if the mouse is inside a Dot that already exists. We do this calculating the distance between the Dot center and the mouse position.

We implement the code of the `mousePressed()` function.

We want to delete an existing Dot when it is pressed, and also to delete all the Dots connected to it, if there are any.

If the distance is less than the radius we delete the current Dot, calling our function `deleteDot()`.

If we delete a Dot we exit from the function with the keyword `return`. This means that the following lines are not executed, and so we do not create a new Dot.