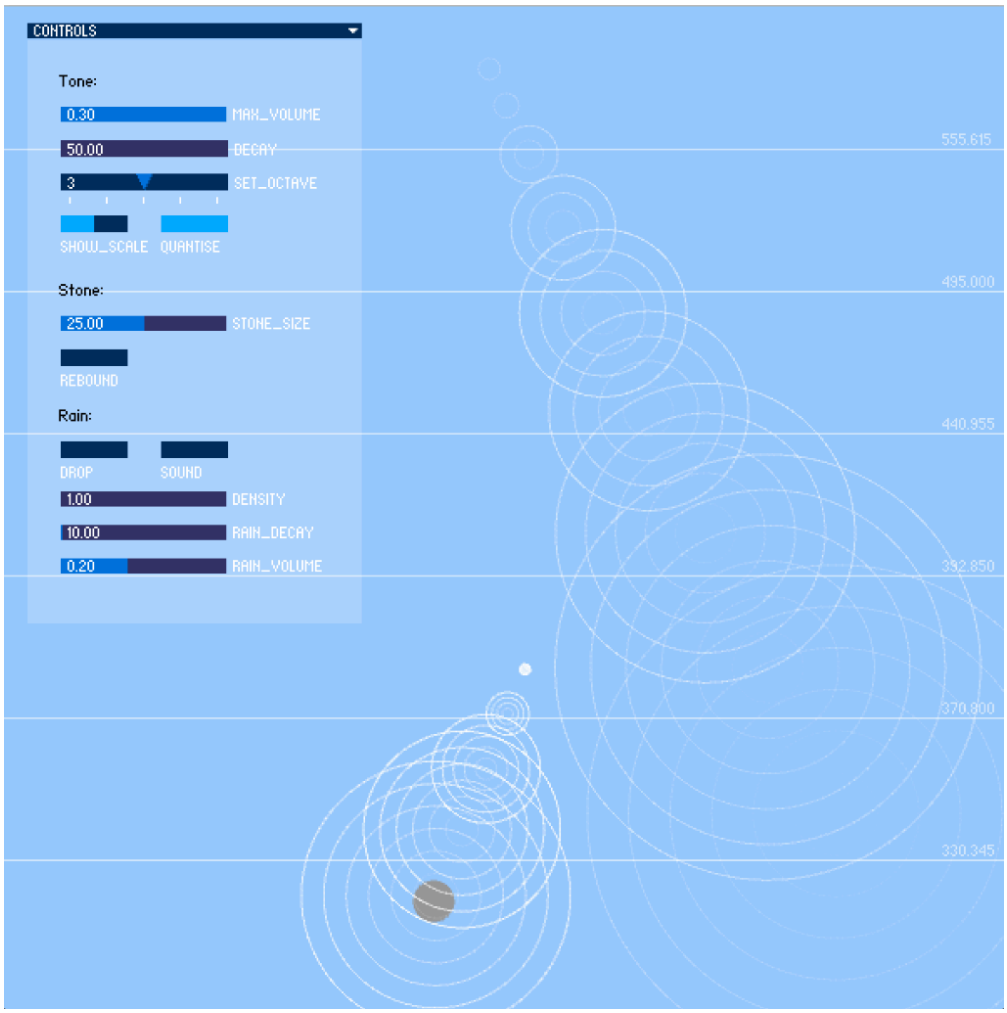
Assignment 3 – Reciprocal Syncresis

Plans Laid Waste

My original idea was to use mouse clicks to create ripples in water, where each ripple would also generate a tone, and as ripples overlapped one another, the tones associated with them would modulate each other's frequencies. However, I had a lot of trouble getting the tone instances to listen to each other.

While working on this problem I found it frustrating having to click each time I wanted to generate a tone, and thought it would be more fun and interesting to create some form of automation. The first idea was a stone skipping across the surface of the pond, where each skip produced a new pitch and ripple.

The second idea, for something a little more stochastic was to introduce rain, where each drop would produce a similar pitched-ripple in the water. To achieve this I created four classes: skipping stone, rain, ripples and tone, which interact with one another in various ways.



Stones
A stone is created each time the mouse is released. Its direction and velocity are derived from the difference between the mouse position when pressed, and when released; this is called the velocity. As the stone travels across the pond it drops in height, when it hits the water (when its height drops below 1) it resets to its maximum height – based on its velocity – which is lowered slightly; meaning it moves shorter distances with each bounce. **Rain** is like an invisible stone that randomly changes height and moves across the screen. When the tones velocity is 0, or when it goes off screen it is deleted.

Ripples
Each time the stone or rain hits the water a new ripple instance is created. The size of the ripple (how many rings are drawn) is derived from the stone's current height reset counter, which gets lower with each bounce; so each bounce will cause smaller and smaller ripples. The ripples grow and fade on each new frame. When the ripples have faded away they are deleted.

Tones
Each time a ripple instance is created, a tone is created with it. The stones x position on screen sets the pitch, it's y position sets the pan, and its current velocity sets the volume. When the tone has gone quiet it is deleted.

((o))

Inspiration
I think a major inspiration was simply learning and writing the code. As I solved lower level problems the way I found solutions helped form new ideas and directions to explore. One example of this was the number of ways to process incoming values, and get related, but different results.

From previous projects and long-term interests, I found it a lot of fun to derive multiple values from a single source. For example, the stone size and sling tension were the root of many other values in the program including the distance it could skip, its size, tone volume, and the size and number of ripples it created.

An earlier example of me exploring these ideas can be found in the link below. In the piece Beroom I took three notes A, B, and D, and their respective frequencies at multiple octaves to control myriad functions across the whole piece including: filter frequency, FM oscillator pitch, oscillator pitch, envelope time, speed of rhythms triggering the envelopes and so on.
http://3490148.tumblr.com/beroom/

I've been aware of some basic processes such as mapping, or rescaling values, inversions and reversals, and transpositions, but what's really starting to draw my attention are different forms of quantisation. I found this badly executed in my code in the form of stone skip distance as related to sling tension, but well implemented in the form of the scale quantisation for tone frequency.

Disasterpeace
While working, trying to find the right music to suit or induce the headspace needed for programming led me repeatedly to Richard Vreeland's work as Disasterpeace for the games Fez, Hyperlight Drifter, and especially The Floor Is Jelly. The tone of these soundtracks was definitely the mood I aimed for in my app, though I'm not sure I got close.
http://music.disasterpeace.com/album/the-floor-is-jelly-ost/

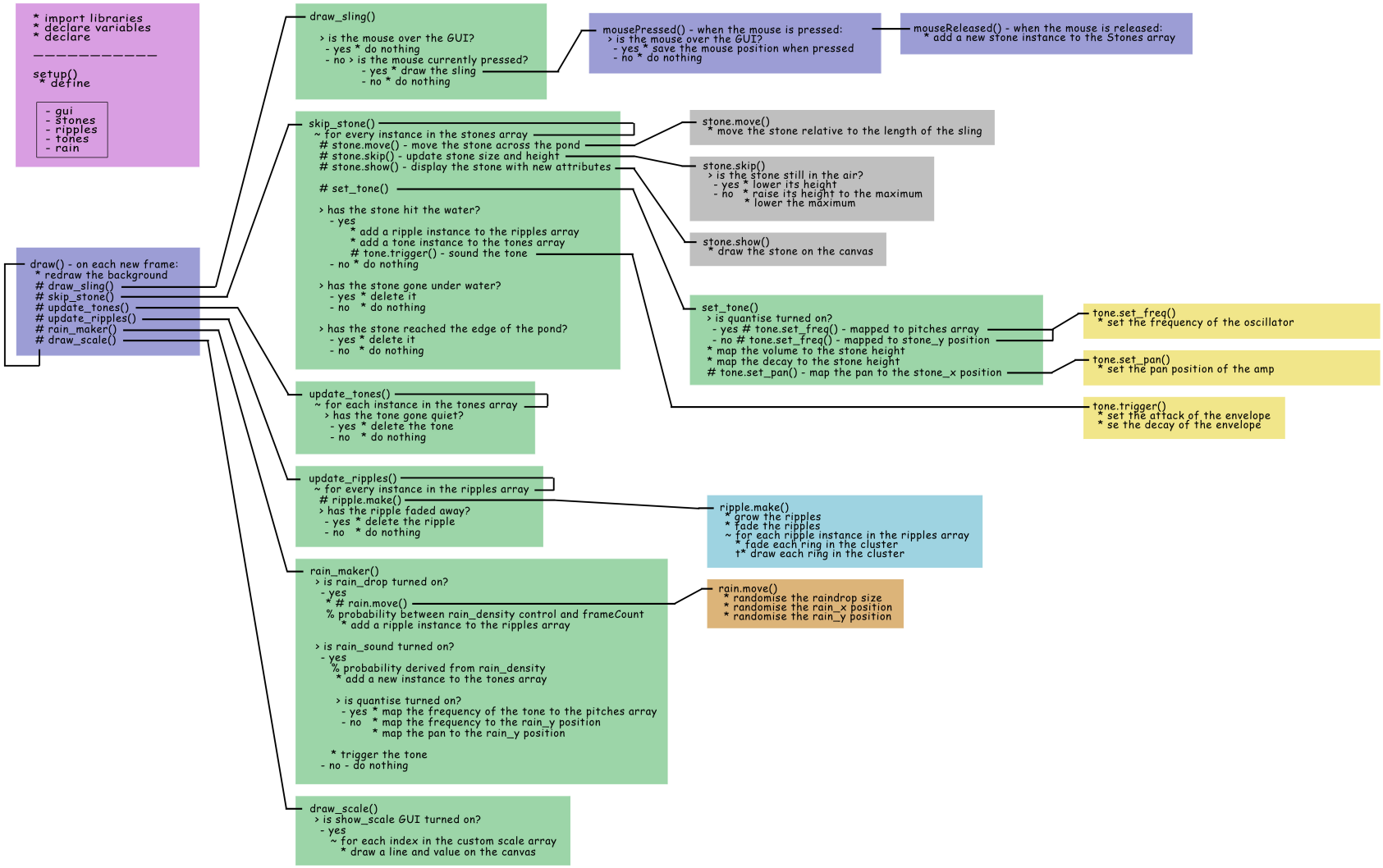
Martine Corompt
Also RMIT's own Martine Corompt, who has been animating water, droplets, rain, and streams by hand for years. I got to catch her piece Torrent, scored by Phillip Brophy, at the Melbourne Centre for Contemporary Photography in 2016. I love the way she is able to express the behaviour of such a dynamic three dimensional medium in just two.
https://vimeo.com/117869216/

((o))

Truncation
If I had the time, skills, or good reason to continue learning Processing (looking like I'll go back to Max, or forward to Open Frameworks right now) there are a few things I would have liked to add to this program:

- The ability to throw different shaped stones that result in different waveforms being played in the oscillator – square, triangle etc.
- Frequency modulation as ripples overlap with one another. My original idea – sharing data between classes worked in tests, but not the main app.
- Change all coordinate systems to vectors. I was really enjoying the vector theory I learned, but couldn't get my head around it fast enough to use.
- Use Perlin noise to create a rippled look to the pond. More an excuse to use Perlin noise – I actually like my super-flat look.
- More sophisticated rain function – possibly playing back filtered samples of actual rain, that interact with the density control and drops on screen.
- Open vs closed pond – the ability to make the stone bounce back from the edges instead of being deleted. I had a version of this working but the stone would stay stuck to the edge of the screen instead of reversing direction due the to the way the stone skip distance was calculated.
- On that note – I would have liked a complete rewrite of the stone skip function, as it works, but not in an easily modifiable way.

Lessons
- Normalised Values - I spent far too much time sending println() commands from functions chasing useable results. I should have used getter pairs after each process providing the actual and normalised output values like we were shown.
- I also had the opportunity (and will try harder in future) to write a reusable class in the form of a tone setter shared by the stone and rain functions, but didn't make it happen; they do share the same tone generator though.


* import libraries
* declare variables
* declare
setup()
* define
- gui
- stones
- ripples
- tones
- rain
draw() - on each new frame:
* redraw the background
# draw_sling()
# skip_stone()
# update_tones()
# update_ripples()
# rain_maker()
# draw_scale()
draw_sling()
> is the mouse over the GUI?
- yes * do nothing
- no > is the mouse currently pressed?
- yes * draw the sling
- no * do nothing
mousePressed() - when the mouse is pressed:
> is the mouse over the GUI?
- yes * save the mouse position when pressed
- no * do nothing
mouseReleased() - when the mouse is released:
* add a new stone instance to the Stones array
skip_stone()
~ for every instance in the stones array
# stone.move() - move the stone across the pond
# stone.skip() - update stone size and height
# stone.show() - display the stone with new attributes
# set_tone()
> has the stone hit the water?
- yes
* add a ripple instance to the ripples array
* add a tone instance to the tones array
# tone.trigger() - sound the tone
- no * do nothing
> has the stone gone under water?
- yes * delete it
- no * do nothing
> has the stone reached the edge of the pond?
- yes * delete it
- no * do nothing
stone.move()
* move the stone relative to the length of the sling
stone.skip()
> is the stone still in the air?
- yes * lower its height
- no * raise its height to the maximum
* lower the maximum
stone.show()
* draw the stone on the canvas
set_tone()
> is quantise turned on?
- yes # tone.set_freq() - mapped to pitches array
- no # tone.set_freq() - mapped to stone_y position
* map the volume to the stone height
* map the decay to the stone height
# tone.set_pan() - map the pan to the stone_x position
tone.set_freq()
* set the frequency of the oscillator
tone.set_pan()
* set the pan position of the amp
tone.trigger()
* set the attack of the envelope
* se the decay of the envelope
update_tones()
~ for each instance in the tones array
> has the tone gone quiet?
- yes * delete the tone
- no * do nothing
update_ripples()
~ for every instance in the ripples array
# ripple.make()
> has the ripple faded away?
- yes * delete the ripple
- no * do nothing
ripple.make()
* grow the ripples
* fade the ripples
~ for each ripple instance in the ripples array
* fade each ring in the cluster
t* draw each ring in the cluster
rain_maker()
> is rain_drop turned on?
- yes
* # rain.move()
% probability between rain_density control and frameCount
* add a ripple instance to the ripples array
> is rain_sound turned on?
- yes
% probability derived from rain_density
* add a new instance to the tones array
> is quantise turned on?
- yes * map the frequency of the tone to the pitches array
- no * map the frequency to the rain_y position
* map the pan to the rain_y position
* trigger the tone
- no - do nothing
rain.move()
* randomise the raindrop size
* randomise the rain_x position
* randomise the rain_y position
draw_scale()
> is show_scale GUI turned on?
- yes
~ for each index in the custom scale array
* draw a line and value on the canvas