Creative Computing CA2

Generative Agency

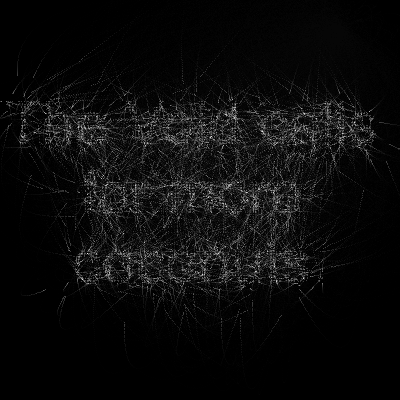
# Project Description

# create an interactive generative time based program that explores agency, shape, movement, type and colour which allows users to interactively change options to suit their visual requirements.

# Project Rationale

With this project I wanted to create a generative agent that would swarm seemingly randomly but eventually come together to form text on the screen. Initially after researching and getting interested in flow fields the idea was to use flowfields or some other particle effect to create agents that would zip around and after a certain amount of time it would become apparent they were forming words



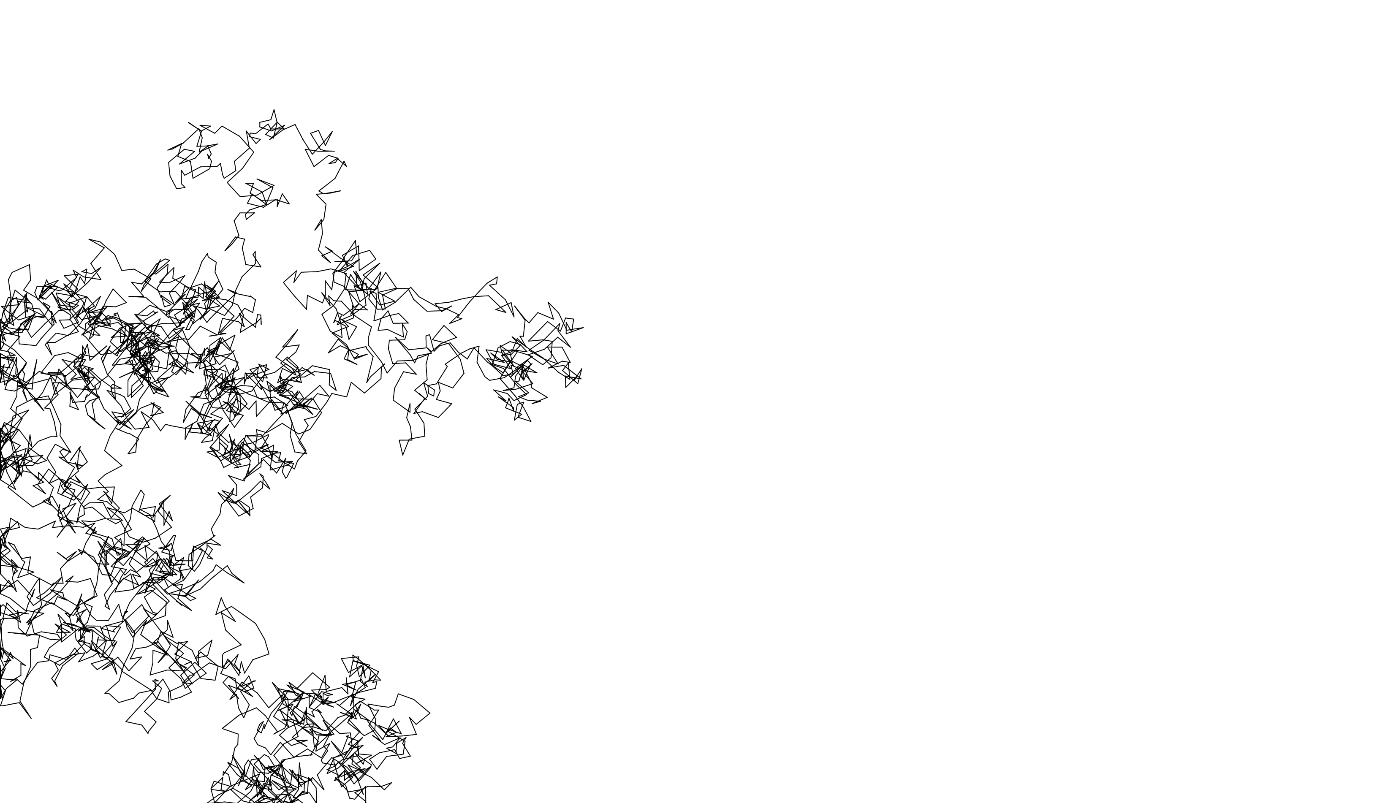


After doing research on the different types of agency and particle physics that would fit within the scope of this project I decided on Brownian motion.

Brownian Motion, named by Benoit Mandelbrot after Robert Brown the botanist who discovered it, is a physical phenomenon of particles moving at random. Simply put it is a particle that moves step by step in any direction and at a random step size which has a set upper limit.

# Implementation

Drawing a line between each step was the first stage of implementing my idea, the following image shows how it looks on the canvas.



The next step in experimenting with this was to see if I could change the step size within the canvas depending on if a particle was over a certain pixel. After some research and remembering what we had been taught I discovered I could save an image to a pixel array. Using this I outlined some pseudo code

// 1.Render a text on the canvas.

// 2.Store the pixels in an array.

// 3.Clear canvas.

// 4.Add Brownian Motion particles.

// 5.Draw a line between the current and the previous position for each particle.

// 6.When a particle is on the text, set the step size small - otherwise big.

To implement the first three steps the following code is used

function myText() {

  var tSize = 150;

  textSize(tSize);

  var tWidth = textWidth(message);

  text(message, w / 2 - tWidth / 2, h / 2 + tSize / 2);

  var image = get(0, 0, w, h);

  image.loadPixels();

  px = image.pixels;

  background(255);

}

This sets the font size, centers the text, prints whatever text is placed in the message variable and then converts the text into an image and saves it in a pixel array name px and finally resets the canvas to blank white.

Particles are then added to the canvas.

function drawParticles() {

  particles = [];

  for(var i = 0; i < 50; i++) {

    particles.push(new Particle());

  }

}

The function creates an empty particles array which is then populated with Particle objects which contain properties to move the particles randomly within the bounds of the step-size

Particle.prototype.move = function(stepSize) {

  this.oldX = this.x;

  this.oldY = this.y;

  this.x += random(-stepSize, stepSize);

  this.y += random(-stepSize, stepSize);

}

The next step is to have the step-size reduce when a particle is over the previously saved pixel array this is achieved using the following if statement check.

var stepSize = 30;

      if(px[off+3] > 100) {

        stepSize = 2;

      }

Which checks if the pixel in the pixel array has a value greater than 100 which every pixel that makes up the word does, then set the step-size to small.

The final stage was to add a gui that added controls for the user to use to be able to manipulate the project easily, these came in the form of being able to change the message and color of the lines being drawn.

var message = 'Creative Coding';

var Color = '#a510ed';

Initial values are set for both optional variables

gui = createGui('Style (Press the DEL key to reset)').setPosition(windowWidth - 220, 20);

  gui.addGlobals('message', 'Color')

The gui is then created and positioned and then the variables are added to the globals which automatically detects the type of input to use based on the value of the variable, so for message a text box is created and for color a color picker is added, one more step must be completed to add opacity to the lines so that they overlay each other and become darker. The color picker created by the gui produces colors in hex codes so to add opacity we take the hex code and simply append the hex value equivalent of the opacity desired, in this case 10% which in hex is ‘1A’

stroke(Color+'1A');

The final output is displayed in the image below.

