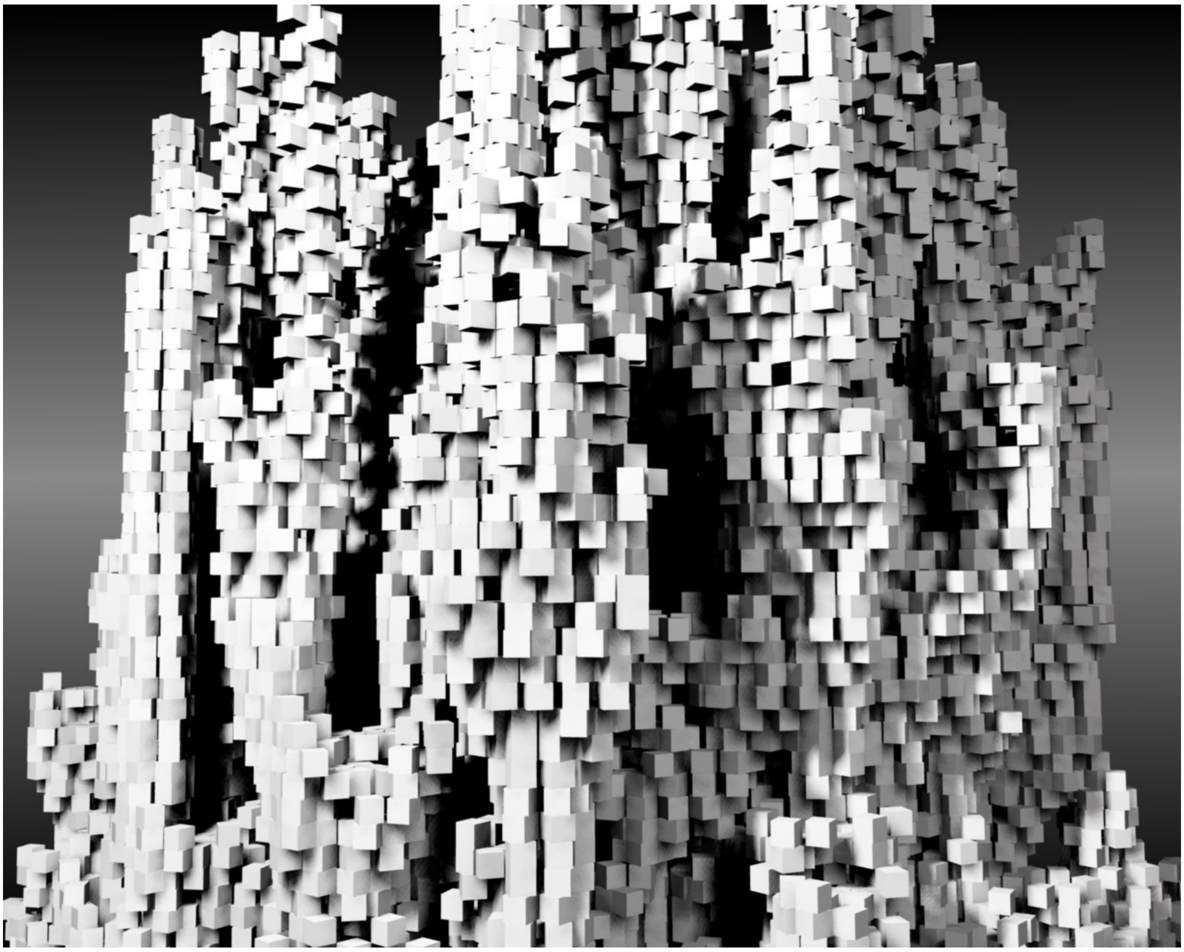
Morphogenetic Programming

Week 1: Cellular Automata



```
// Simple Wolfram CA

int row = 1;
color black = color(0);

// rule 110
// 2 + 4 + 8 + 32 + 64 = 110
int [] response = { 0, 1, 1, 1, 0, 1, 1, 0 };

void setup()
{
  size(200,400);
  set(0,0,black);
}

void draw()
{
  for (int i = 0; i < width; i++)
  {
    color a = get( (i+width-1) % width, row-1);
    color b = get(i,row-1);
    color c = get( (i+width+1) % width, row-1);

    int number = 0;

    if (a == black) {
      number += 1;
    }
    if (b == black) {
      number += 2;
    }
    if (c == black) {
      number += 4;
    }

    if (response[number] == 1) {
      set(i,row,black);
    }
  }
  row++;
}
```

```
// Conway's game of life glider

boolean [][] cells;
boolean [][] nextcells;

void setup()
{
  size(400,400);
  cells = new boolean [100][100];
  nextcells = new boolean [100][100];
  for (int i = 0; i < 100; i++) {
    for (int j = 0; j < 100; j++) {
      cells[i][j] = false;
    }
  }
  // glider setup
  cells[50][50] = true;
  cells[49][50] = true;
  cells[51][50] = true;
  cells[51][49] = true;
  cells[50][48] = true;
  frameRate(6);
}

void draw()
{
  background(192);
  for (int i = 0; i < 100; i++) {
    for (int j = 0; j < 100; j++) {
      if (cells[i][j] == true) {
        rect(i*4,j*4,4,4);
      }
      // count up how many alive around us
      int number = 0;
      for (int m = -1; m <= 1; m++) {
        for (int n = -1; n <= 1; n++) {
          if (!(m == 0 && n == 0)) {
            if (cells[(i+m+100)%100][(j+n+100)%100] == true) {
              number++;
            }
          }
        }
      }
      // counted up... now set state of cell for next generation
      if (number < 2 || number > 3) {
        nextcells[i][j] = false;
      }
      else if (number == 3) {
        nextcells[i][j] = true;
      }
      else {
        nextcells[i][j] = cells[i][j];
      }
    }
  }
  for (int i = 0; i < 100; i++) {
    for (int j = 0; j < 100; j++) {
      cells[i][j] = nextcells[i][j];
    }
  }
}
```

```
// Idealised Belousov–Zhabotinsky reaction

// An implementation note about this algorithm is available here:
// http://www.aac.bartlett.ucl.ac.uk/processing/samples/bzr.pdf

float [][][] a;
float [][][] b;
float [][][] c;

int p = 0, q = 1;

void setup()
{
  size(400,400);
  colorMode(HSB,1.0);
  a = new float [width][height][2];
  b = new float [width][height][2];
  c = new float [width][height][2];
  for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
      a[x][y][p] = random(0.0,1.0);
      b[x][y][p] = random(0.0,1.0);
      c[x][y][p] = random(0.0,1.0);
      set(x,y,color(0.5,0.7,a[x][y][p]));
    }
  }
}

void draw()
{
  for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
      float c_a = 0.0;
      float c_b = 0.0;
      float c_c = 0.0;
      for (int i = x - 1; i <= x+1; i++) {
        for (int j = y - 1; j <= y+1; j++) {
          c_a += a[(i+width)%width][(j+height)%height][p];
          c_b += b[(i+width)%width][(j+height)%height][p];
          c_c += c[(i+width)%width][(j+height)%height][p];
        }
      }
      c_a /= 9.0;
      c_b /= 9.0;
      c_c /= 9.0;
      // adjust these values to alter behaviour
      a[x][y][q] = constrain(c_a + c_a * (c_b - c_c), 0, 1);
      b[x][y][q] = constrain(c_b + c_b * (c_c - c_a), 0, 1);
      c[x][y][q] = constrain(c_c + c_c * (c_a - c_b), 0, 1);
      set(x,y,color(0.5,0.7,a[x][y][q]));
    }
  }
  if (p == 0) {
    p = 1; q = 0;
  }
  else {
    p = 0; q = 1;
  }
}
```