# Los Algoritmos Genéticos y su Aplicación al Arte Generativo

Autor: Emiliano Causa | 2011

emiliano.causa@gmail.com

## Resumen

En este trabajo se aborda la forma de aplicar Algoritmos Genéticos a trabajos de Arte Generativo, atendiendo a la forma en que debe ser generado el proceso de herencia. Durante el trabajo se verá un caso en que esta técnica se aplica a una aplicación capaz de generar cuadros con la estética de Victor Vasarely. Posteriormente se tratará otro caso en que la técnica se aplica a imágenes fractales. En este último ejemplo se profundizará en su implementación a nivel de programación. Por último se discutirán los criterios para la representación de los genes en el proceso.

*algoritmos genéticos*

*crossover*

*genes*

*Victor Vasarely*

*fractales*

## Introducción

El presente texto es una nueva versión mi texto “Vasarely Genético”, escrito en 2003. Por esto, grandes partes de este texto están extraídas del anterior, pero en la actual versión intento corregir algunas nociones erróneas del anterior, así como dar a luz cuestiones técnicas implicadas en este tipo de proceso. Por otra parte, en este trabajo, no se basa exclusivamente en el trabajo “Vasarely Genético” sino que aborda nuevos ejemplos y expande la problemática a cuestiones más universales.

## Antecedentes en el arte genético

“La naturaleza utiliza potentes medios para impulsar la evolución satisfactoria de los organismos. Los organismos que son poco aptos para un determinado ambiente mueren, en tanto que los que están bien adaptados para vivir, se reproducen. Los hijos son semejantes a sus padres, por lo que cada nueva generación tiene organismos semejantes a los miembros bien dotados de la generación anterior.” (Russel y Norvig, 1996)

Los algoritmos genéticos son una técnica de la Inteligencia Artificial, que simula el proceso evolutivo de los seres vivos y lo aplica a la búsqueda de soluciones y optimización, en la resolución de problemas. El arte genético es generado por computadora a partir de algoritmos genéticos. Ejemplos de la aplicación de los algoritmos genéticos la podemos encontrar en Genetic Images de Karl Sims, “es una instalación multimedia en la que los visitantes pueden interactuar en el proceso evolutivo de imágenes abstractas. Una supercomputadora genera y muestra imágenes en 16 pantallas situadas en el espacio en forma de arco. Los visitantes se paran sobre sensores frente a las imágenes que les resultan de mayor belleza, y así seleccionan las imágenes que sobrevivirán y se reproducirán en una nueva generación” (Sims,1993). “Genetic Images, Particle Dreams y Panspermia, obras de Karl Sims, junto con The Process Of The Evolution de Will Lantham o Mutations de Yoichiro Kawaguchi” (Macchi, 2002), son obras de arte genético que fueron expuestas en el Centro Georges Pompidou durante la Revue Virtuelle de 1993.

## Vasarely Genético

Durante el 2003 desarrollé, en mi trabajo dentro del grupo Proyecto Biopus – www.biopus.com.ar-, un trabajo llamado Vasarely Genético. El objetivo del proyecto fue realizar una obra interactiva de arte genético para la participación colectiva (a través de Internet u otros medios) basada en cuadros de arte-óptico generados por computadora, que imitan la estética de Victor Vasarely. Los usuarios podían recorrer una colección de cuadros, y seleccionar dos, para que se reproduzcan genéticamente. La reproducción daba como resultado un tercer cuadro que heredaba y combinaba las

características de sus progenitores. Dicho procedimiento hacía que la colección evolucione según el gusto de los usuarios. En las siguientes páginas, utilizaré este trabajo para ejemplificar la aplicación de algoritmos genéticos al arte (las técnicas del arte genético).

## El proceso evolutivo

¿Puede una obra de arte (o un fenómeno estético cualquiera) evolucionar por sus propios medios de la misma forma que lo hacen los seres vivos? El proceso evolutivo de los seres vivos, nos fascina por su capacidad de auto-construirse, auto-modificarse y encontrar el diseño adecuado para cada medio ambiente, diseño no falto de belleza ni variedad. Intuyo que esa característica es la que nos invita a copiarlo y aplicarlo al arte, para que nuestras obras evolucionen independientes de nosotros, sus autores. Los algoritmos genéticos son las técnicas que nos permiten simular el proceso evolutivo y aplicarlo a sistemas artificiales.

En este trabajo, el sistema en cuestión es una colección de pinturas de arte óptico. Para lograr la simulación, los elementos de nuestro sistema tienen que comportarse, con la lógica del proceso evolutivo, como si fueran actores dispuestos a representar el juego de la evolución. Para que esto suceda, debemos decidir que papel le toca a cada actor. ¿Cuáles son nuestros actores y sus papeles? Los protagonistas de este trabajo son los cuadros (las pinturas), y a ellos les toca el papel de individuos. La colección es la población. ¿Quién sería entonces el medio ambiente, a cargo de la selección natural? o dicho de otra forma ¿según qué criterio debe evolucionar esta población? El público hará de medio ambiente y a través de sus elecciones realizará la selección natural. Repartidos los papeles, nos queda profundizar en dos procesos: el de la reproducción genética (que nos permite heredar las características de los progenitores y combinarlas para potenciarlas) y el proceso de selección natural que permite que sólo sobrevivan los aptos.

## La reproducción genética y la herencia

En la reproducción se encuentran dos individuos que combinan su material genético para heredar sus características y combinarlas. ¿Cómo se realiza esto? El material genético de cada individuo está en el cromosoma, organizado en una colección de genes, durante la reproducción se toma al azar, genes de uno u otro progenitor (a esta operación se la denomina crossover) y se construye el nuevo material genético mezclando los cromosomas. Por lo tanto, para poder reproducir los cuadros, debemos preguntarnos ¿cuál es su material genético? y ¿cómo se organiza este material en genes? Estas dos preguntas son el tema central de este trabajo.

## El material genético

En el material genético (el genotipo) están escritas las instrucciones para construir el individuo (el fenotipo), por lo tanto, para poder representarlo en los términos de los algoritmos genéticos, en este material genético debería estar escrito el plan de construcción de un cuadro con la estética de Vasarely. ¿Cuáles son las instrucciones para construir un cuadro con esta estética? Para responder esta pregunta fue necesario realizar un análisis de los cuadros. Cabe aquí una aclaración, para la realización de este trabajo se analizaron 8 pinturas (de los primeros períodos del artista), y por tanto la estética de este trabajo se restringe a este pequeño conjunto y no a la basta obra de este artista; sin embargo descubriremos que este pequeño conjunto posee un gran potencial.

El análisis dio el siguiente resultado: Los cuadros están conformados por una grilla de cuadrados, en donde en cada cuadrado existe una figura interna más chica (que puede ser cuadrada o circular), a su vez, esta figura puede poseer (no siempre se da) una interior (también cuadrada o circular), todas concéntricas con la figura a la que pertenecen. A la grilla la llamaremos primer estrato, a las figuras interiores, segundo estrato, y las interiores a estas, tercer estrato. Cada figura posee un color que se continúa en las figuras del mismo estrato, variando paulatinamente, generando un degradé que posee una dirección y un grado de variación. Las figuras de un mismo estrato tienen la misma figura, o cuadrado o círculo. La configuración que hasta aquí hemos descripto es lo que denominamos lienzo inicial, el cual manifiesta continuidad en cada uno de sus estratos. El resultado se hace interesante cuando se combinan todos los estratos con los distintos degradé y figuras. La figura 1 muestra los tres estratos por separados, (A) el primero, (B) el segundo, (C) el tercero, y el cuadro completo (D). En ella se puede observar la dirección de cada degradé y la forma en que se combinan.

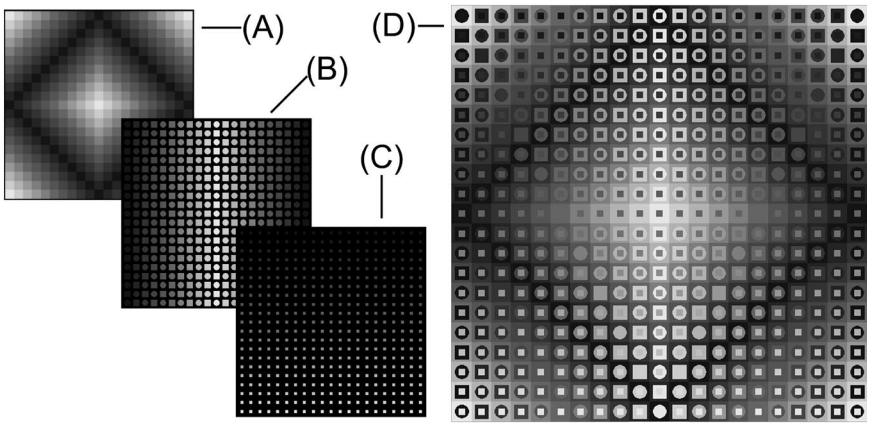
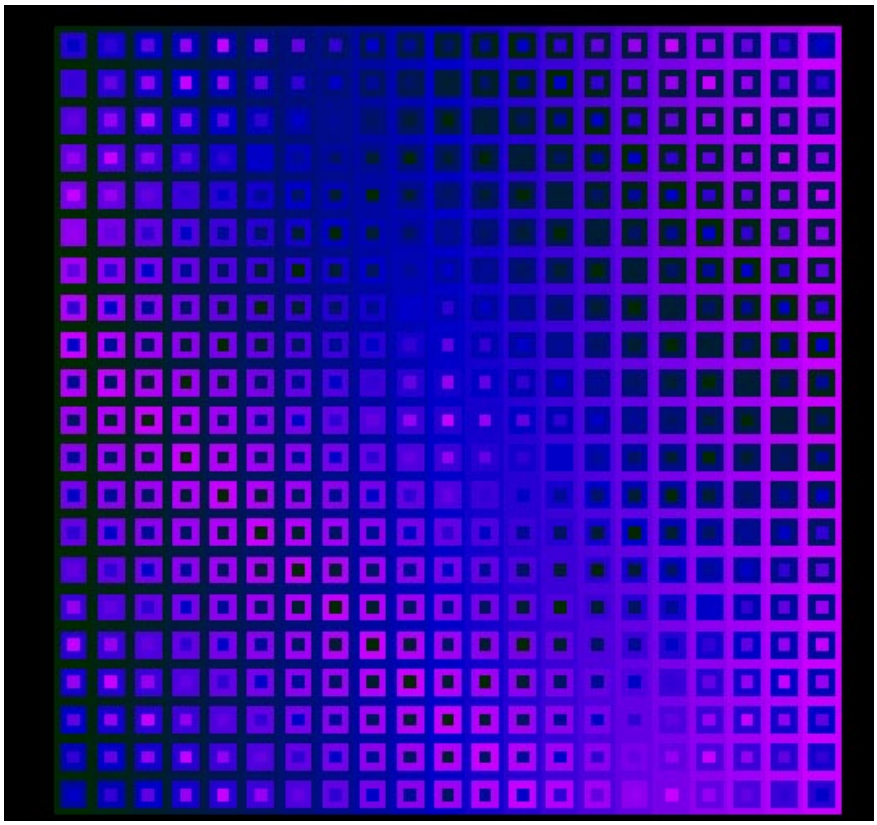
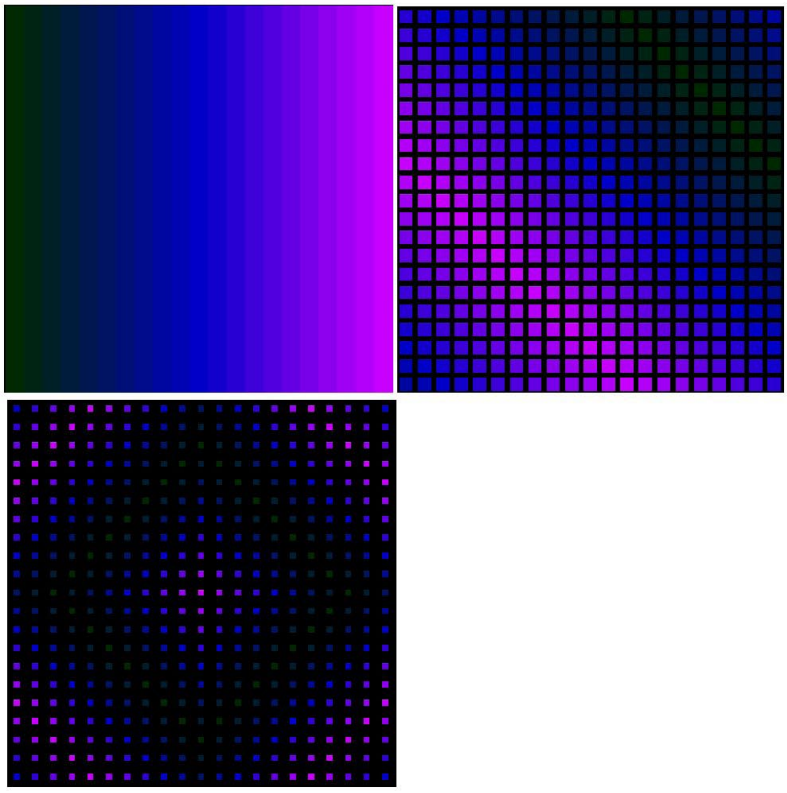



*Figura 1*

En la figura 1 se observa que en el segundo estrato se alternan (en forma de damero) círculos y cuadrados. Esto es debido a que la configuración establecida en un estrato puede ser variada dentro áreas de cambio. Estas áreas de cambio pueden ser dameros o

formas simétricas (axiales). En la figura 2 se aplicaron dos áreas de cambio al ejemplar de la figura 1, la primera divide el lienzo por la mitad y la segunda es de forma cuadrada. En estas se alteraron el degradé y el tipo de figuras de los estratos.





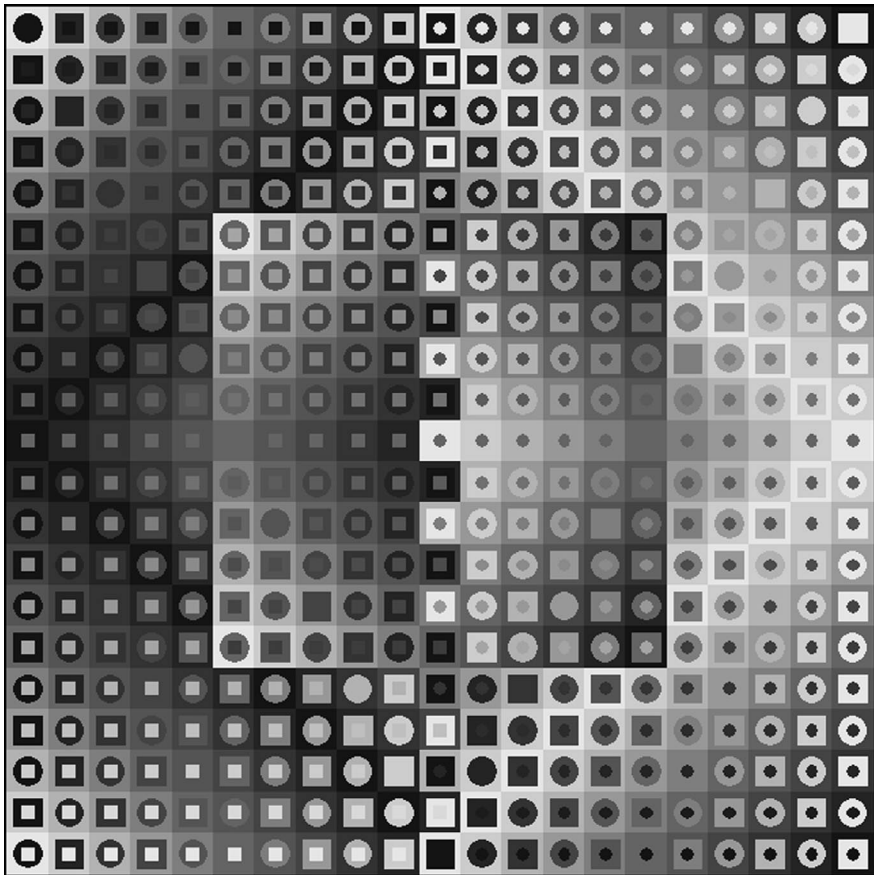


Figura 2

En las siguientes imágenes podemos ver otro ejemplo de un cuadro conformado por estratos, la primer figura es el cuadro completo y debajo se puede ver el despiece en estratos:

Las imágenes también pueden poseer zonas de cambio, en donde se producen alteraciones respecto de los degradé o del tipo de figura que conforman el estrato. Esto puede verse en las siguiente imagen, en las que se pueden visualizar algunas “áreas de cambio”: la primera divide el cuadro por la mitad con un corte vertical y la segunda es una cuadrado en el centro de la imagen.

En el interior de un área de cambio se generan alteraciones que pueden ser de distintos tipos: cambio de color, cambio de dirección o grado del degradé, cambio de figura. Las superposiciones de áreas de cambio enriquecen las posibilidades de variación del lienzo inicial, dado que combinan sus alteraciones. Por lo tanto, el material genético de los cuadros debe tener la información necesaria para definir: 1)un lienzo inicial con sus propiedades: tamaño de la grilla, cantidad de estratos, color, degradé y figura de cada estrato, 2)las áreas de cambio con sus propiedades de tamaño y ubicación, 3) y los cambios a realizar en cada área.

## Organización genética

Definido el material genético, abordamos entonces nuestra segunda pregunta: ¿cómo se organiza este material en genes? Esta cues-

tión tiene especial importancia, dado que esta organización incide directamente sobre la capacidad de heredar. Si deseamos que la herencia se manifieste en el fenotipo, es decir que el hijo muestre rasgos similares a sus padres, entonces debemos distinguir cuáles son los rasgos más notables de un cuadro, para así organizar los cromosomas de forma de preservarlos. La organización en genes es una formación de compromiso entre la capacidad de combinar cualidades y la necesidad de mantener inalterados ciertos rasgos: cuánto mayor es la cantidad de genes, mayor será la variedad posible en la reproducción (existen más combinaciones realizables), sin embargo la relación con sus progenitores será cada vez más ambigua, llegando al límite de no distinguirse la herencia de ningún rasgo; por la inversa, si la cantidad de genes es poca y éstos engloban muchos rasgos, entonces la herencia será manifiesta en el fenotipo, pero habrá poco lugar para la innovación.

Buscando una solución intermedia y luego de varias pruebas, definí 5 genes, correspondientes a lo que creo los rasgos más sobresalientes de los cuadros (el orden no tiene relevancia):

Gen 1: Tamaño de la grilla y áreas de cambio

Gen 2: Dirección y grado de los degradé

Gen 3: Alteraciones producidas en las áreas de cambio

Gen 4: Figuras del lienzo inicial

Gen 5: Colores

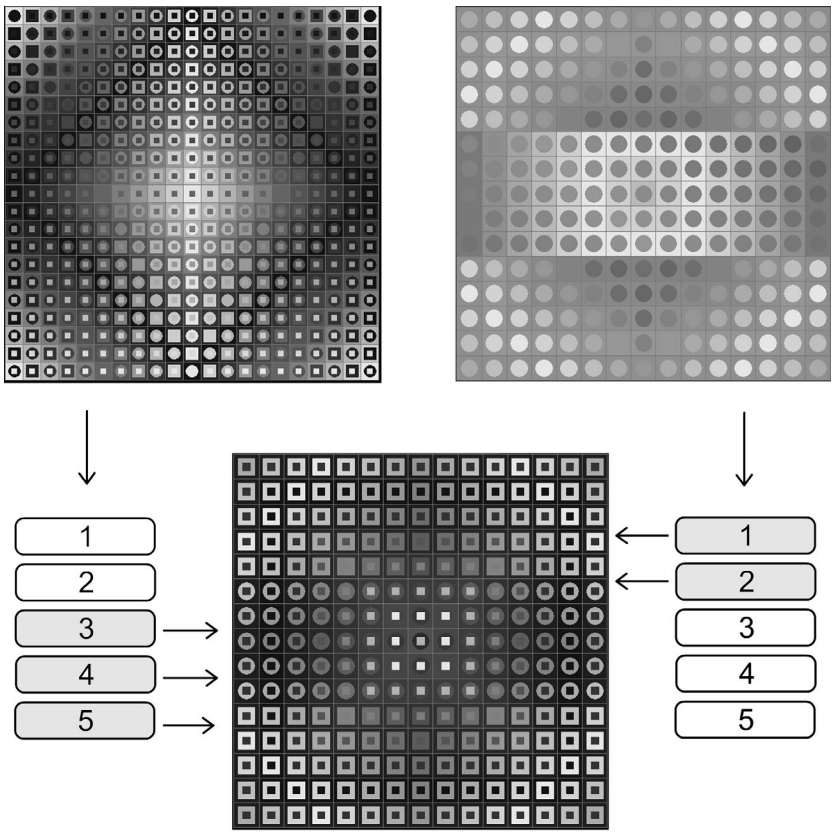



*Figura 3*

En la figura 3 puede observarse, a los cuadros progenitores (arriba), el cuadro resultante de la reproducción (abajo), el cual tomó los 2 primeros genes del progenitor izquierdo y el resto del de la derecha.

## La composición de los genes

¿De qué se componen los genes? Cuando trabajamos con Algoritmos Genéticos los genes de los "sujetos" que cruzamos están compuestos de los valores de las variables que permiten construir al sujeto en cuestión. Es decir, los valores de las variables que caracterizan al sujeto. Pero cuando aplicamos Algoritmos Genéticos a un fenómeno estético, como un cuadro con la estética de Vasarely, es importante que los genes reflejen propiedades perceptibles. Los cinco genes elegidos en nuestro ejemplo, responden a características que pueden ser "vistas". Para ésto, fue necesario englobar diversas variables en cada uno de los genes. Si se hubiera hecho una asociación de un gen por variable, seguramente la cantidad de genes superaría los treinta, y la cantidad de combinaciones posibles sería altísima, pero dado que no todas las variables representan características que son perceptibles por sí mismas, seguramente en la reproducción se perdería la herencia de las características percibidas de los progenitores. Y es que se puede distinguir entre las características constructivas y las perceptibles, es decir, las variables aisladas definen la forma en que se construye el cuadro, pero para poder definir una característica perceptible, es necesario reunir varias de éstas en un sólo gen. El criterio bajo el cuál se realiza la reunión de variables, es mediante la percepción en sí, es decir, es necesario observar el fenómeno resultante de la

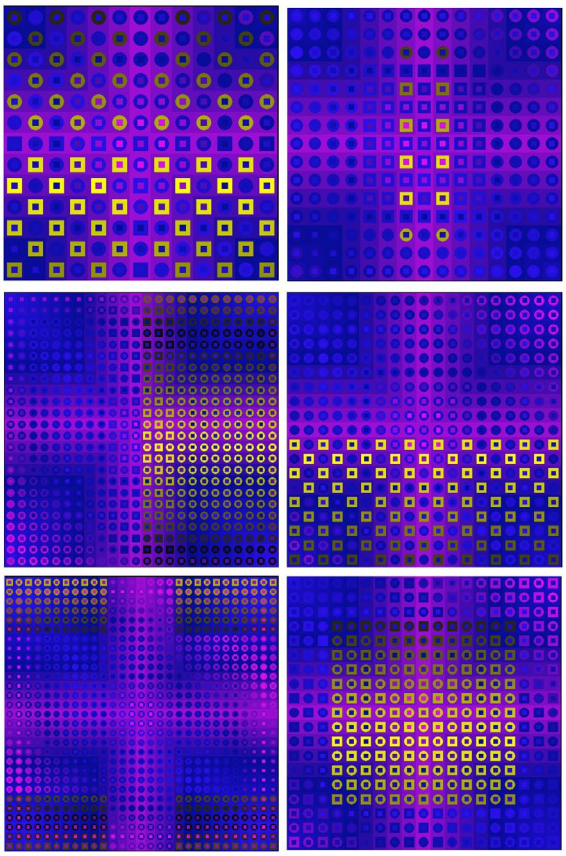


*Figura: Variaciones del Gen 1: Tamaño de la grilla y áreas de cambio*

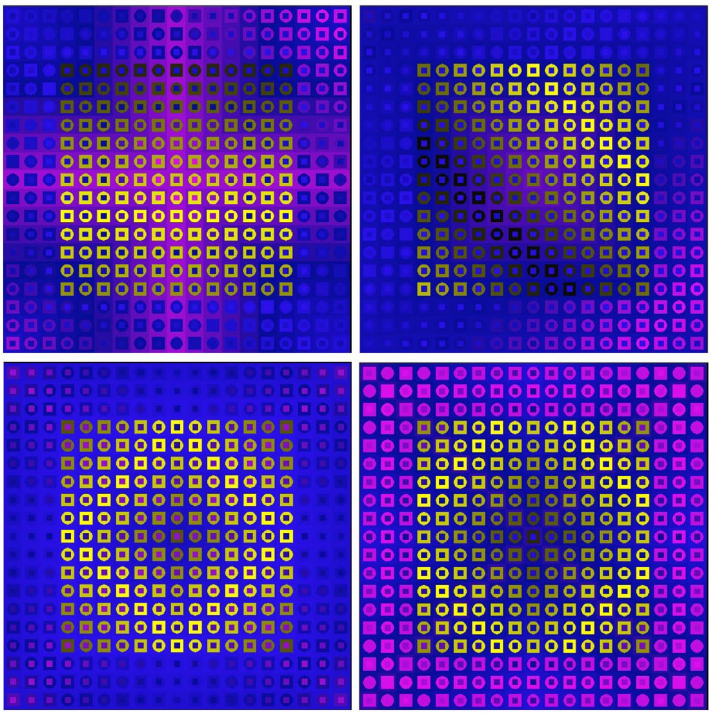


*Figura: Variaciones del Gen 2: Dirección y grado de los degradé*

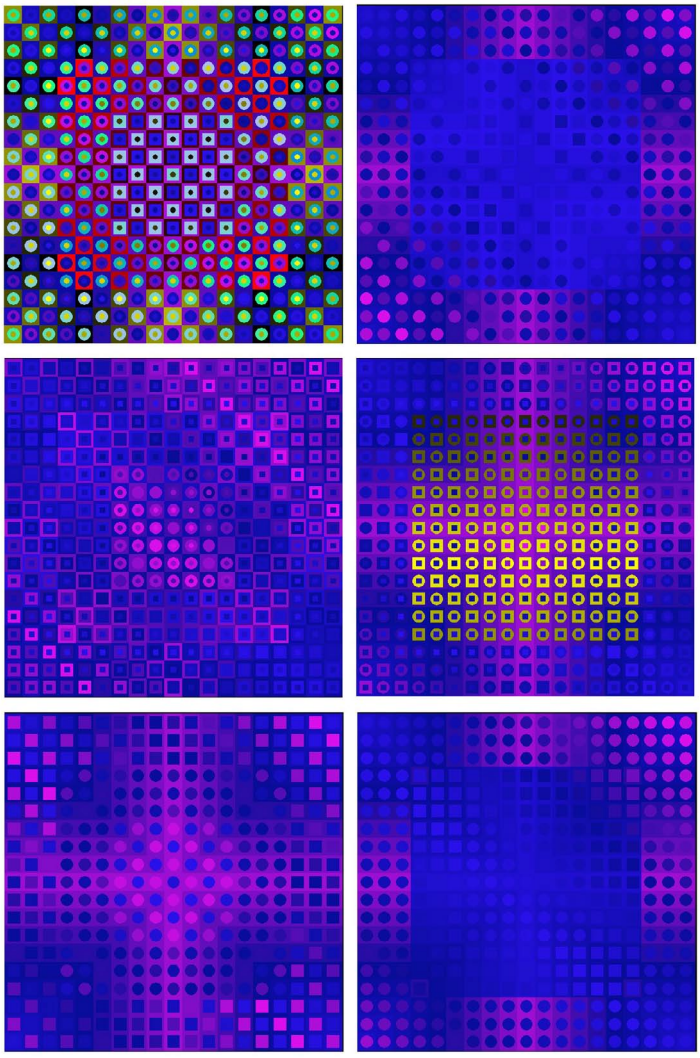


*Figura: Variaciones del Gen 3: Alteraciones producidas en las áreas de cambio*

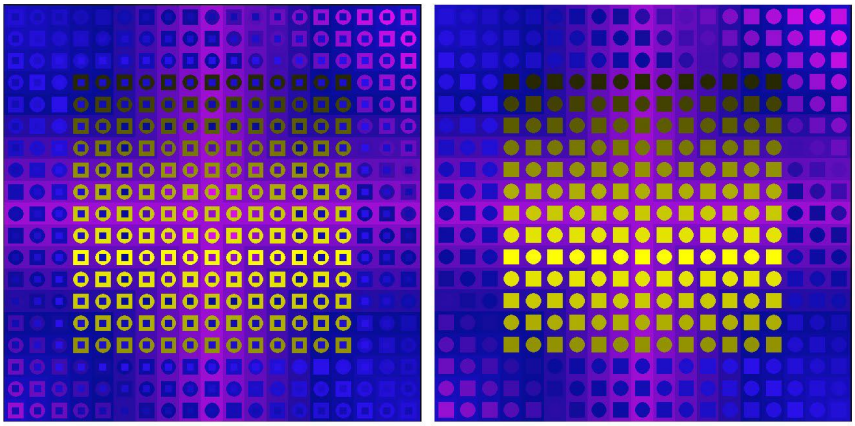


*Figura: Variaciones del Gen 4: Figuras del lienzo inicial*

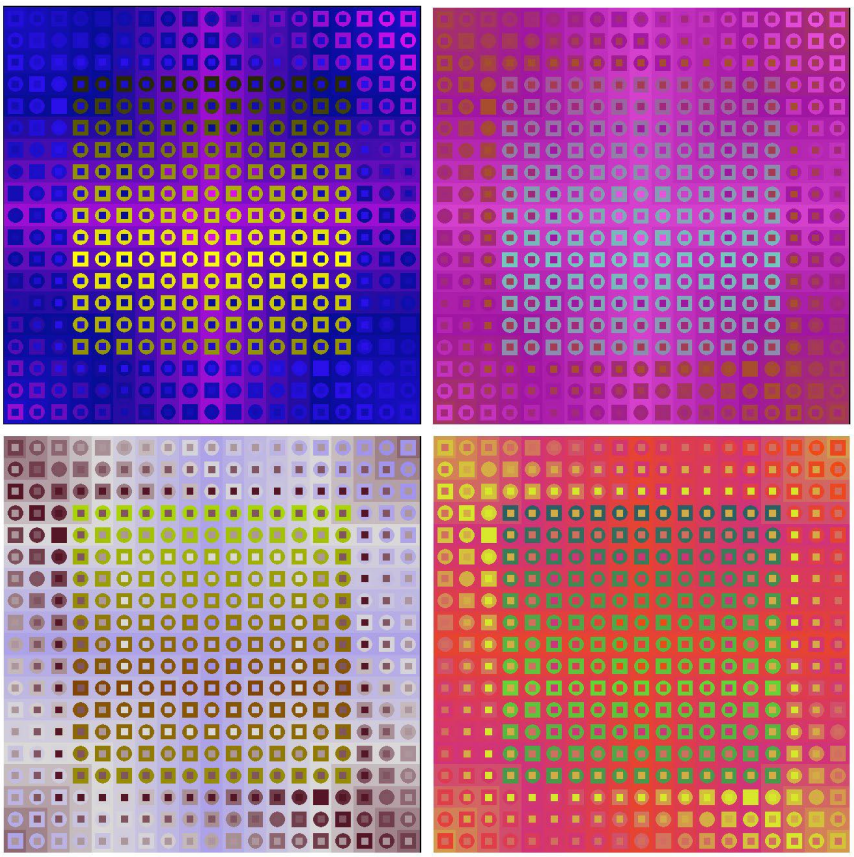
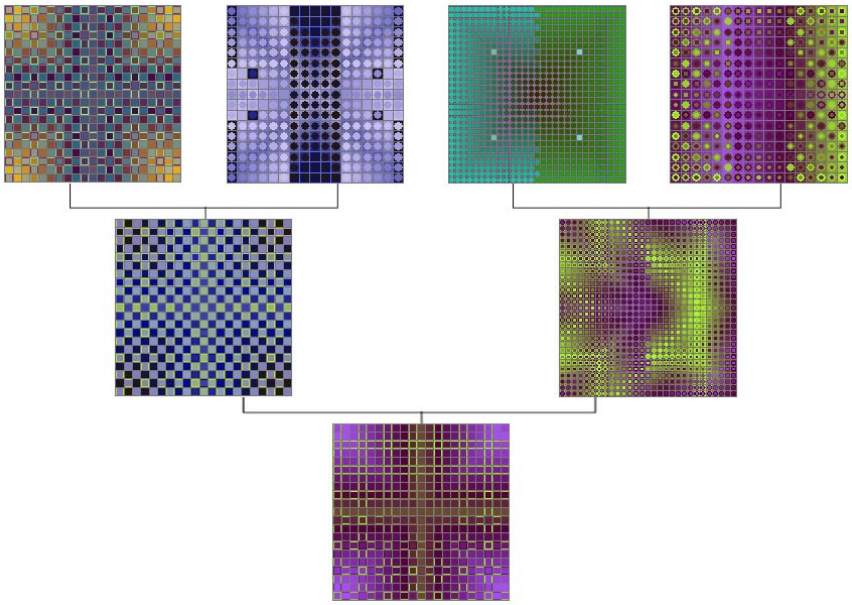


*Figura: Variaciones del Gen 5: Colores*

construcción y tratar de analizar estas características.

En las siguientes imágenes veremos variaciones sobre cada uno de los genes.

Como se puede ver en la figuras anteriores, cada Gen representa una característica fácilmente perceptible y por ende sus combinaciones en la reproducción por cross-over producirá la herencia de dichos caracteres bajo patrones también perceptibles. En la figura siguiente podemos ver la evidencia de dicha herencia en el árbol genealógico de una cuadro. La primer hilera representa a los “abuelos” del cuadro, la segunda a los hijos de los abuelos (los “padres”) y debajo, finalmente, al cuadro resultante de todos estas



generaciones:

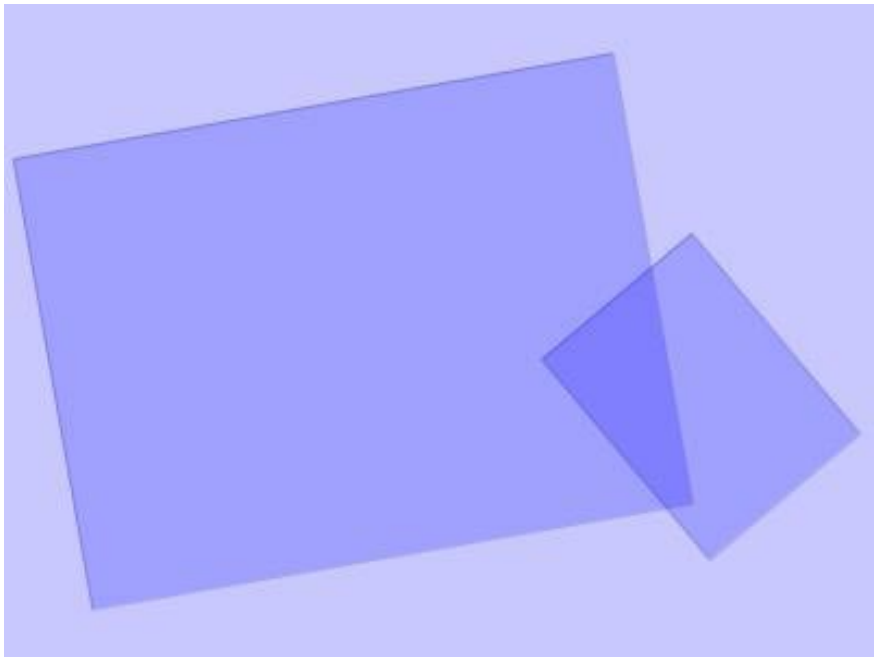
En este caso, el color del nieto viene por los cuadros ubicados más a la derecha, y las figuras por la rama ubicada más a la izquierda.

## Implementación del crossover

El ejemplo que tomaremos ahora adopta la generación de una imagen fractal mediante funciones recursivas, el procedimiento y algoritmo para realizar dicha figura han sido tratados en otro capítulo, por lo que no nos extenderemos con ello en el presente. Sólo tomaremos el caso, ya en funcionamiento, y lo adaptaremos para ser utilizado con algoritmos genéticos. El trabajo está desarrollado con el lenguaje Processing (www.processing.org).

En el texto sobre fractales, se vio una función recursiva para crear fractales con rectángulos, esta función toma datos de cuatros arreglos que poseen las propiedades de X, Y, ángulo y factor de amortiguación. En las siguiente imagen pueden verse los rectángulos generados en una sola iteración para los siguientes valores:

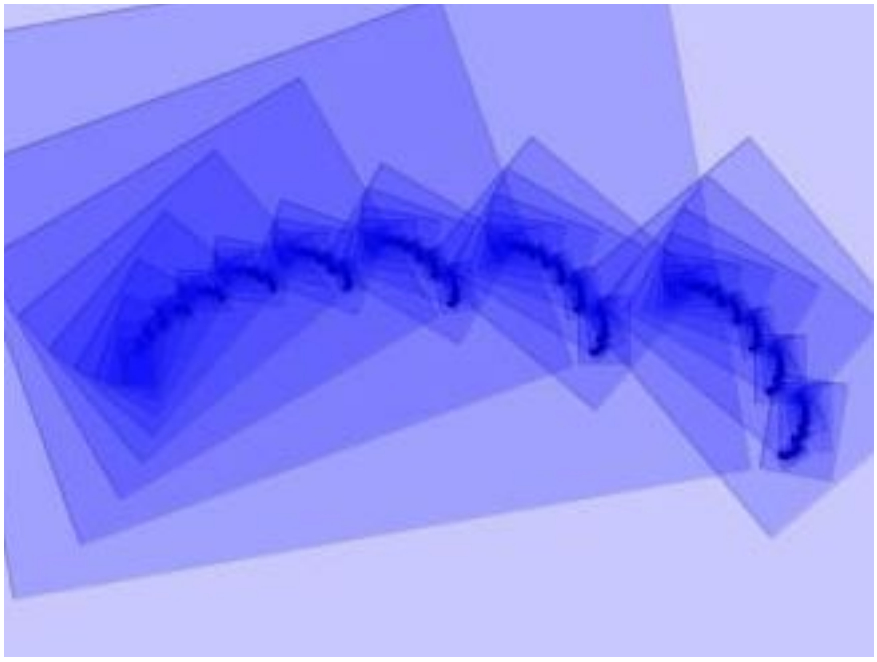
```
float factor[] = { 0.7 , 0.3 };
float x[] = { 0.4 , 0.8 };
float y[] = { 0.5 , 0.6 };
float ang[] = { radians(-10) , radians(50) };
```



Es decir, el primer rectángulo esta ubicado en la posición ( 0.4 , 0.5 ), teniendo en cuenta que el ancho de  todo el marco va hasta 1. El ángulo es de -10 grados y el tamaño respecto del marco es 0.7. El segundo cuadrado tiene coordenadas ( 0.8 , 0.6 ), ángulo de 50 grados y tamaño de 0.3.

En la siguiente imagen se ve el fractal que se produce cuando se hacen varias iteraciones recursivas.

Ahora veremos una adaptación, en la que la función recursiva es



ejecutada dentro de una clase llamada Fractal. En el Anexo 1 se encuentra una transcripción completa de la clase. Para nuestros fines en el presente texto, repasaremos el constructor de la clase, que es donde se inicializan las propiedades principales de la clase. A continuación podemos ver el título del constructor con los parámetros:

```
Fractal( int ancho_ , int alto_ , int cantidad_ , int nivel,
 float factor_[], float x_[], float y_[], float ang_[],
 float iniTinte_ , float iniSatura_ , float iniBrillo_ ,
 float incTinte_ , float incSatura_ , float incBrillo_ ,
 float transparencia_ ,
 float CiniTinte_ , float CiniSatura_ , float CiniBrillo_ ,
 float CincTinte_ , float CincSatura_ , float CincBrillo_ ,
 float Ctransparencia_
   ) {
```

Detalle de los parámetros:

int ancho_ = ancho del marco

int alto_ = alto del marco

int cantidad_ = cantidad de rectángulos

int nivel = niveles de iteraciones recursivas

float factor_[] = tamaños de los rectángulos respecto de su marco (factor de multiplicación)

float x_[] = posiciones en X de los rectángulos

float y_[] = posiciones en Y de los rectángulos

float ang_[] = ángulos de rotación de los rectángulos

float iniTinte_ = tinte inicial del color de relleno

float iniSatura_ = saturación inicial del color de relleno

float iniBrillo_ = brillo inicial del color de relleno

float incTinte_ = incremento en cada iteración del tinte del color de relleno

float incSatura_ = incremento en cada iteración de la saturación del color de relleno

float incBrillo_ = incremento en cada iteración del brillo del color de relleno

float transparencia_ = transparencia del color de relleno

float CiniTinte_ = tinte inicial del color de contorno

float CiniSatura_ = saturación inicial del color de contorno

float CiniBrillo_ = brillo inicial del color de contorno

float CincTinte_ = incremento en cada iteración del tinte del color de contorno

float CincSatura_ = incremento en cada iteración de la saturación del color de contorno

float CincBrillo_ = incremento en cada iteración del brillo del color de contorno

float Ctransparencia_ = transparencia del color de contorno

Además de los parámetros que describimos al principio, en la clase existen parámetros destinados a definir el color de relleno y de contorno. Estos colores están definidos con una paleta HSB (Hue Saturation Brightness, Tinte/Saturación/Brillo) que resulta más intuitiva y rica para controlar. Por cada uno de los dos colores (relleno y contorno) se define los valores iniciales de Tinte, Saturación y Brillo y luego se define cuánto se incrementa en cada iteración.

A la hora de poder trabajar con estos objetos usando algoritmos genéticos, debemos acceder a aquellas variables que definen las propiedades de estos fractales. Queda claro que esas variables son los parámetros que acabamos de detallar, pero para poder disponer de ella en una forma genética haremos algunos cambios.

Lo más útil es poder disponer de la información genética sin tener que arrastrar a la clase completa para ello. Por eso, lo mejor es poner todos los parámetros en un único arreglo, para que este cumpla la función de una cadena de ADN. De esta forma, la clase Fractal se comporta como el "fenotipo" y el arreglo con los parámetros, como el "genotipo". Así el fenotipo se construye a partir de la información recibida con el genotipo. La forma en que se realiza esto es haciendo un nuevo constructor que recibe como parámetro al arreglo que contiene a todos los parámetros:

```
Fractal( int ancho_, int alto_, float adn[] ) {

  float factor_[] = new float[3];
  float x_[] = new float[3];
  float y_[] = new float[3];
  float ang_[] = new float[3];

  arrayCopy( adn, 2, factor_, 0, 3 );
  arrayCopy( adn, 5, x_, 0, 3 );
  arrayCopy( adn, 8, y_, 0, 3 );
  arrayCopy( adn, 11, ang_, 0, 3 );

  inicio( ancho_, alto_, int( adn[0] ), //cantidad

int( adn[1] ), //nivel
  factor_, x_, y_, ang_,
  adn[14], //iniTinte
  adn[15], //iniSatura
  adn[16], //iniBrillo_ ,
```

```
  adn[17], //incTinte _ ,
   adn[18], //incSatura _ ,

  adn[19], // incBrillo _ ,
   adn[20], //transparencia _ ,
   adn[21], //CiniTinte _ ,
   adn[22], //CiniSatura _ ,
   adn[23], //CiniBrillo _ ,
   adn[24], //CincTinte _ ,
   adn[25], //CincSatura _ ,
   adn[26], // CincBrillo _ ,
   adn[27]  //Ctransparencia _
   );

 }
```

En este caso, el constructor recibe el arreglo ( que aquí se llama adn[ ] ) y utiliza cada una de sus celdas de memoria para asignar valor a los parámetros de una función inicio( ), que cumple las veces del constructor original y recibe los mismos parámetros. Lo importante aquí es lograr que todos los parámetros estén en una única estructura de datos, un arreglo, para poder hacer el crossover (es decir: el entre-cruzamiento de las porciones de ADN de los progenitores para la construcción de la cadena del hijo). De esta forma se puede manipular la estructura genética sin necesidad de lidiar con los objetos, y sólo usar la clase y los objetos cuando hace falta producir los fractales. Por ejemplo, a continuación se puede ver como se inicializan los parámetros del objeto del tipo Fractal mediante la asignación de valores a un arreglo llamado “adn”. De esta forma el arreglo es literalmente una “cadena” de información, tal como lo es el ADN:

```
Fractal objetoFractal;
float adn[] = {
  2 , 5 , // cantidad , nivel
  0.7 , 0.1 , 0.4 , //factores de tamaño
  0.2 , 0.75 , 0.8 , //posiciones en x
  0.5 , 0.5 , 0.5 , //posiciones en y
  radians(-10) , radians(0) , radians(15) , //angulos
  60,255,255, //HSB inicial relleno
  10,0,0 ,  //HSB incremento relleno
  180, // transparencia
  0,0,0, //HSB inicial contorno
  30,2,10, //HSB incremento contorno
  255 //transparencia contorno
};
objetoFractal = new Fractal( 300 , 200 , adn );
```

Una vez logrado este pasaje de los parámetros a una cadena independiente del objeto, podemos hacer la operación de crossover simplemente operando en las cadenas, es decir en los arreglos que tienen el ADN. A continuación se puede ver la función de crossover, la cual recibe como parámetros las cadenas de ADN del padre y de la madre, representados por las clases p[ ] y m[ ], respectiva-

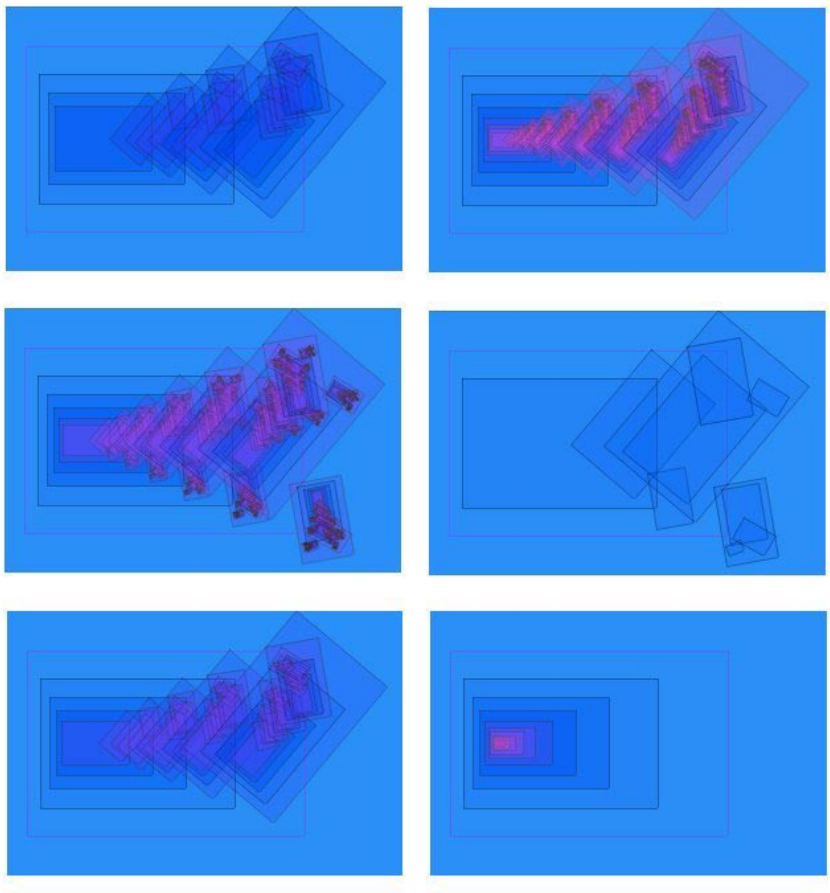
mente. Esta función recibe dos arreglos y, por supuesto, devuelve un arreglo con la combinación de sus genes. Es decir que construyen un genotipo combinando dos que recibe:

```
float[] crossOver( float p[] , float m[] ){
  float nuevoAdn[] = new float[28];

  if( random(100)<50 ){ //GEN 1: cantidad y niveles
    nuevoAdn[0] = p[0];
    nuevoAdn[1] = p[1];
  }
  else{
    nuevoAdn[0] = m[0];
    nuevoAdn[1] = m[1];
  }

  if( random(100)<50 ){ //GEN 2: factor,x,y,ang del rectangulo 1
    nuevoAdn[2] = p[2];
    nuevoAdn[5] = p[5];
    nuevoAdn[8] = p[8];
    nuevoAdn[11] = p[11];
  }
  else{
    nuevoAdn[2] = m[2];
    nuevoAdn[5] = m[5];
    nuevoAdn[8] = m[8];
    nuevoAdn[11] = m[11];
  }

  if( random(100)<50 ){ //GEN 3: factor,x,y,ang del rectangulo 2
    nuevoAdn[3] = p[3];
    nuevoAdn[6] = p[6];
    nuevoAdn[9] = p[9];
    nuevoAdn[12] = p[12];
  }
  else{
    nuevoAdn[3] = m[3];
    nuevoAdn[6] = m[6];
    nuevoAdn[9] = m[9];
    nuevoAdn[12] = m[12];
  }

  if( random(100)<50 ){ //GEN 4: factor,x,y,ang del rectangulo 3
    nuevoAdn[4] = p[4];
    nuevoAdn[7] = p[7];

   nuevoAdn[10] = p[10];
    nuevoAdn[13] = p[13];
  }
  else{
    nuevoAdn[4] = m[4];

   nuevoAdn[7] = m[7];
    nuevoAdn[10] = m[10];

   nuevoAdn[13] = m[13];
```

```
}


if( random(100)<50 ){ //GEN 5: colores de relleno

 nuevoAdn[14] = p[14];
  nuevoAdn[15] = p[15];
  nuevoAdn[16] = p[16];
  nuevoAdn[17] = p[17];
  nuevoAdn[18] = p[18];
  nuevoAdn[19] = p[19];
}
else{
  nuevoAdn[14] = m[14];
  nuevoAdn[15] = m[15];
  nuevoAdn[16] = m[16];
  nuevoAdn[17] = m[17];
  nuevoAdn[18] = m[18];
  nuevoAdn[19] = m[19];
}

if( random(100)<50 ){ //GEN 6: transparencia de relleno
  nuevoAdn[20] = p[20];
}
else{
  nuevoAdn[20] = m[20];
}

if( random(100)<50 ){ //GEN 7: colores del contorno
  nuevoAdn[21] = p[21];
  nuevoAdn[22] = p[22];
  nuevoAdn[23] = p[23];
  nuevoAdn[24] = p[24];
  nuevoAdn[25] = p[25];
  nuevoAdn[26] = p[26];
}
else{
  nuevoAdn[21] = m[21];
  nuevoAdn[22] = m[22];
  nuevoAdn[23] = m[23];
  nuevoAdn[24] = m[24];
  nuevoAdn[25] = m[25];
  nuevoAdn[26] = m[26];
}

if( random(100)<50 ){ //GEN 8: transparencia de relleno
  nuevoAdn[27] = p[27];
}
else{

 nuevoAdn[27] = m[27];
}

return nuevoAdn;
```

```
}
```

En esta función podemos observar como se agrupan los diferentes parámetros en cada uno de los genes. Por ejemplo: en el Gen 1 se agruparon las celdas 0 y 1 del arreglo, las cuales corresponden a la cantidad de rectángulos y a los niveles (cantidad de iteraciones). En las siguientes imágenes se pueden ver variaciones sobre el Gen 1.



La forma en que se realiza la agrupación de los parámetros en el Gen 1 es asignándolos juntos en la selección al azar entre los genes del padre y de la madre:

```
if( random(100)<50 ){ //GEN 1: cantidad y niveles
  nuevoAdn[0] = p[0];
  nuevoAdn[1] = p[1];
}
else{
  nuevoAdn[0] = m[0];
  nuevoAdn[1] = m[1];
}
```

La selección se hace con un 50% de probabilidades para cada una de las posibilidades, gracias al procedimiento de obtener un número Random entre 0 y 100, para luego separar aquellos que son

menores que 50 de aquellos que no lo son.

Como puede verse en la función, los genes agrupan los siguientes parámetros:

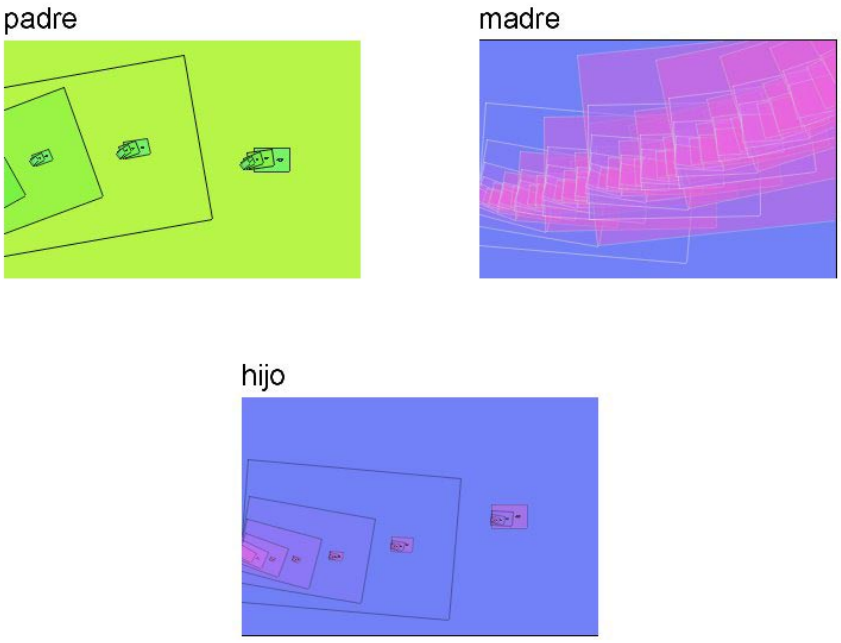
```
GEN 1: cantidad rectángulos y niveles de iteración
       parámetros 0 y 1
GEN 2: tamaño, x, y, ángulo del rectangulo 1
       parámetros 2, 5, 8 y 11
GEN 3: tamaño, x, y, ángulo del rectangulo 2
       parámetros 3, 6, 9 y 12
GEN 4: tamaño, x, y, ángulo del rectangulo 3
       parámetros 4, 7, 10 y 13
GEN 5: colores de relleno
       parámetros 14, 15, 16, 17, 18 y 19
GEN 6: transparencia de relleno
       parámetro 20
GEN 7: colores del contorno
       parámetros 21, 22, 23, 24, 25 y 26
GEN 8: transparencia de relleno
       parámetro 27
```

Una vez que la función de crossover está resuelta de esta forma, se puede hacer la siguiente operación:

```
Fractal padre,madre,hijo;
float adnPadre[] = {
  2 , 5 , // cantidad , nivel
  0.7 , 0.1 , 0.4 , //factores de tamaño
  0.2 , 0.75 , 0.8 , //posiciones en x
  0.5 , 0.5 , 0.5 , //posiciones en y
  radians(-10) , radians(0) , radians(15) , //angulos
  60,255,255, //HSB inicial relleno
  10,0,0 ,  //HSB incremento relleno
  180, // transparencia
  0,0,0, //HSB inicial contorno
  30,2,10, //HSB incremento contorno
  255 //transparencia contorno
};
float adnMadre[] = {
  2 , 6 , // cantidad , nivel
  0.6 , 0.8 , 0.75 , //factores de tamaño
  0.3 , 0.7 , 0.8 , //posiciones en x
  0.6 , 0.4 , 0.5 , //posiciones en y
  radians(5) , radians(-6) , radians(10) , //angulos
  170,155,255, //HSB inicial relleno
  10,10,0 ,  //HSB incremento relleno
  100, // transparencia
  0,0,255, //HSB inicial contorno
  30,2,-10, //HSB incremento contorno
  100 //transparencia contorno
};
padre = new Fractal( 300 , 200 , adnPadre );
madre = new Fractal( 300 , 200 , adnMadre );
```

```
float adnHijo[] = crossOver( adnPadre , adnMadre );
hijo = new Fractal( 300 , 200 , adnHijo );
```

El código escrito arriba es una simplificación en el que se omitieron pasos que no están directamente relacionados con la problemática. En el Anexo 2 está la versión completa del programa. Independiente de esto, en la anteúltima línea se puede ver como se carga el arreglo adnHijo[ ] con los datos resultantes de la combinación del ADN del




padre y la madre, mediante la función de crossover. En la figura debajo puede observarse como se produce la herencia de carácteres de dos fractales a través de la función de crossover.

## Conclusión

Los algoritmos genéticos son una técnica muy potente que permite simular ciertos aspectos de la reproducción de los seres vivos y aplicarla a diferentes problemáticas, como es el caso del arte, nuestro objeto de discusión en el presente texto. A la hora de aplicar esta técnica a la producción estética, es importante tener en cuenta que el principal interés en su uso radica en la herencia de caracteres y que por tanto, dicha herencia debe ser notable (perceptible) ya que sino la técnica sólo funcionará como una configuración al azar. Si la aplicación de los algoritmos genéticos se da sobre procesos generativos, como han sido los ejemplos expuestos, entonces es necesario extraer cuales son los parámetros que definen la configuración final del fenómeno y luego agrupar estos parámetros de forma tal que representen características perceptibles del fenómeno.

Una estrategia que suele facilitar el procedimiento es hacer que

todos los parámetros sean de un mismo tipo de dato, como en el último ejemplo en donde todos los parámetros son de tipo numérico. Ésto permite concatenarlos en un arreglo y procesarlos como "cadenas de información" sin prestar atención a su contenido. De hecho, en una cadena de ADN, toda la información se reduce a cadenas conformadas por cuatros letras químicas A,C,G y T. Cuando la aplicación de tipos numéricos para todos los parámetros se hace inviable, he tomado como solución transformar todos los datos a cadenas de caracteres (String), para su tratamiento en forma homogénea, luego, el objeto en cuestión se encarga de transformar los tipos de datos.

Espero que el presente texto haya servido para echar luz sobre los procedimientos para aplicar algoritmos genéticos al arte. Existen infinidad de detalles de la técnica pendientes de ser tratados con mayor profundidad, lo que excede a este texto introductorio y que espero tratar en futuros trabajos.

Emiliano Causa

Septiembre 2011

## Bibliografía


- Macchi, Carlos.:2000, Lo vivo en el arte genético, Ars e verse, http://www.arseverse.com/
- Russel Stuart y Norvig Peter.: 1996, Inteligencia Artificial: un enfoque moderno, Prentice Hall, México, pp. 653-654.
- Sims, Karl.:1993, Genetic Images, Karl Sims home page, http://www.genarts.com/karl/

# Anexo 1 – clase Fractal

```
class Fractal {
  PGraphics imagen;

  int anchoLienzo, altoLienzo;
  int cantidad;
  float factor[];
  float x[];
  float y[];
  float ang[];

  float iniTinte;
  float iniSatura;
  float iniBrillo;
  float incTinte;
  float incSatura;
  float incBrillo;
  float transparencia;

  float CiniTinte;
  float CiniSatura;
  float CiniBrillo;
  float CincTinte;
  float CincSatura;
  float CincBrillo;
  float Ctransparencia;

  Fractal( int ancho_ , int alto_ , float adn[] ) {

    float factor_[] = new float[3];
    float x_[] = new float[3];
    float y_[] = new float[3];
    float ang_[] = new float[3];

    arrayCopy( adn, 2, factor_, 0, 3 );
    arrayCopy( adn, 5, x_, 0, 3 );
    arrayCopy( adn, 8, y_, 0, 3 );
    arrayCopy( adn, 11, ang_, 0, 3 );

    inicio( ancho_ , alto_ , int( adn[0] ), //cantidad
    int( adn[1] ), //nivel
    factor_ , x_ , y_ , ang_ ,
    adn[14], //iniTinte
    adn[15], //iniSatura
    adn[16], //iniBrillo_ ,
    adn[17], //incTinte_ ,
    adn[18], //incSatura_ ,
    adn[19], // incBrillo_ ,
    adn[20], //transparencia_ ,
```

```
    adn[21], //CiniTinte _ ,
    adn[22], //CiniSatura _ ,
    adn[23], //CiniBrillo _ ,
    adn[24], //CincTinte _ ,
    adn[25], //CincSatura _ ,
    adn[26], // CincBrillo _ ,
    adn[27]   //Ctransparencia _
    );
  }

  Fractal( int ancho _ , int alto _ , int cantidad _ , int nivel,
  float factor _ [], float x _ [], float y _ [], float ang _ [],
  float iniTinte _ , float iniSatura _ , float iniBrillo _ ,
  float incTinte _ , float incSatura _ , float incBrillo _ ,

 float transparencia _ ,
  float CiniTinte _ , float CiniSatura _ , float CiniBrillo _ ,
  float CincTinte _ , float CincSatura _ , float CincBrillo _ ,
  float Ctransparencia _
    ) {

    inicio( ancho _ , alto _ , cantidad _ , nivel,
    factor _ , x _ , y _ , ang _ ,
    iniTinte _ , iniSatura _ , iniBrillo _ ,
    incTinte _ , incSatura _ , incBrillo _ ,
    transparencia _ ,
    CiniTinte _ , CiniSatura _ , CiniBrillo _ ,
    CincTinte _ , CincSatura _ , CincBrillo _ ,
    Ctransparencia _
      );
  }

  void inicio( int ancho _ , int alto _ , int cantidad _ , int nivel,
  float factor _ [], float x _ [], float y _ [], float ang _ [],
  float iniTinte _ , float iniSatura _ , float iniBrillo _ ,
  float incTinte _ , float incSatura _ , float incBrillo _ ,
  float transparencia _ ,
  float CiniTinte _ , float CiniSatura _ , float CiniBrillo _ ,
  float CincTinte _ , float CincSatura _ , float CincBrillo _ ,
  float Ctransparencia _
    ) {

    anchoLienzo = ancho _ ;
    altoLienzo = alto _ ;

    cantidad = cantidad _ ;

    factor = factor _ ;
    x = x _ ;
    y = y _ ;
    ang = ang _ ;

    iniTinte = iniTinte _ ;
    iniSatura = iniSatura _ ;
    iniBrillo = iniBrillo _ ;
```

```
    incTinte = incTinte_;
    incSatura = incSatura_;
   incBrillo = incBrillo_;

    transparencia = transparencia_;

    CiniTinte = CiniTinte_;
    CiniSatura = CiniSatura_;
    CiniBrillo = CiniBrillo_;

    CincTinte = CincTinte_;
    CincSatura = CincSatura_;
    CincBrillo = CincBrillo_;

    Ctransparencia = Ctransparencia_;

    imagen = createGraphics( anchoLienzo, altoLienzo, P2D );

    imagen.beginDraw();
    imagen.smooth();
    imagen.colorMode(HSB);
    imagen.background( iniTinte, iniSatura, iniBrillo );

    ejecutar( nivel,
    anchoLienzo,
    altoLienzo,
    iniTinte,
    iniSatura,
    iniBrillo,
    CiniTinte,
    CiniSatura,
    CiniBrillo
      );

    imagen.endDraw();
  }

  void ejecutar( int nivel,
  float ancho,
  float alto,
  float tinte,
  float satura,
  float brillo,
  float Ctinte,
  float Csatura,
  float Cbrillo
    ) {

   if ( nivel>0 ) {
       imagen.rectMode( CENTER );
       imagen.fill( tinte, satura, brillo, transparencia );
       imagen.stroke( Ctinte, Csatura, Cbrillo, Ctransparencia
);
```

```
      float proxTinte = ( tinte+incTinte ) % 256;
      float proxSatura = ( satura+incSatura ) % 256;
      float proxBrillo = ( brillo+incBrillo ) % 256;

      float CproxTinte = ( Ctinte+CincTinte ) % 256;
      float CproxSatura = ( Csatura+CincSatura ) % 256;
      float CproxBrillo = ( Cbrillo+CincBrillo ) % 256;

      for ( int i=0 ; i<cantidad ; i++ ) {
        imagen.pushMatrix();
        imagen.translate( ancho*x[i], alto*y[i] );
        imagen.rotate( ang[i] );
        imagen.rect( 0, 0, ancho*factor[i], alto*factor[i] );
);    imagen.translate( ancho*factor[i]*-0.5, alto*factor[i]*-0.5

        pushStyle();
        ejecutar( nivel-1, ancho*factor[i], alto*factor[i],
        proxTinte, proxSatura, proxBrillo,
        CproxTinte, CproxSatura, CproxBrillo
          );
        popStyle();

        imagen.popMatrix();
      }
    }
  }

  void dibujar( float x, float y ) {
    image( imagen, x, y );
  }

  void string() {
    println("cantidad="+cantidad);
    println( "-factor---------" );
    println( factor );
    println( "-x---------" );
    println( x );
    println( "-y---------" );
    println( y );
    println( "-ang---------" );
    println( ang );
    println(  "iniTinte="+iniTinte);
    println(  "iniSatura="+iniSatura);
    println(  "iniBrillo="+iniBrillo);
    println(  "incTinte="+incTinte);
    println(  "incSatura="+incSatura);
    println(  "incBrillo="+incBrillo);
    println(  "transparencia="+transparencia);
    println(  "CiniTinte="+CiniTinte);
    println(  "CiniSatura="+CiniSatura);
    println(  "CiniBrillo="+CiniBrillo);
    println(  "CincTinte="+CincTinte);
    println(  "CincSatura="+CincSatura);
    println(  "CincBrillo="+CincBrillo);
    println(  "Ctransparencia="+Ctransparencia);
```

```
  }

}
```

# Anexo 2 - Cuerpo principal de la aplicación de cruce genético

```
Fractal padre,madre,hijo;
PFont fuente;
/*
  adnPadre, adnMadre y adnHijo son arreglos de tipo flotante
que tienen cifrado
 los genes de cada una de las imágenes, poseen 28 celdas cada uno.
 */
float adnPadre[] = {
  2 , 5 , // cantidad , nivel
  0.7 , 0.1 , 0.4 , //factores de tamaño
  0.2 , 0.75 , 0.8 , //posiciones en x
  0.5 , 0.5 , 0.5 , //posiciones en y
  radians(-10) , radians(0) , radians(15) , //angulos
  60,255,255, //HSB inicial relleno
  10,0,0 ,  //HSB incremento relleno
  180, // transparencia
  0,0,0, //HSB inicial contorno
  30,2,10, //HSB incremento contorno
  255 //transparencia contorno
};

float adnMadre[] = {
  2 , 6 , // cantidad , nivel
  0.6 , 0.8 , 0.75 , //factores de tamaño
  0.3 , 0.7 , 0.8 , //posiciones en x
  0.6 , 0.4 , 0.5 , //posiciones en y
  radians(5) , radians(-6) , radians(10) , //angulos
  170,155,255, //HSB inicial relleno
  10,10,0 ,  //HSB incremento relleno
  100, // transparencia
  0,0,255, //HSB inicial contorno
  30,2,-10, //HSB incremento contorno
  100 //transparencia contorno
};
void setup(){
  size(800,600);
  fuente = loadFont("arial.vlw");
  textFont( fuente , 24 );
  //aquí se les pasa a cada uno de los fractales sendas cadenas de adn
  padre = new Fractal( 300 , 200 , adnPadre );
  madre = new Fractal( 300 , 200 , adnMadre );

  // la función crossOver() se encarga de hacer
  // una nueva cadena de ADN, combinando
  // los ADN del padre y la madre
  float adnHijo[] = crossOver( adnPadre , adnMadre );
  hijo = new Fractal( 300 , 200 , adnHijo );
```

```
}
void draw(){
  background(0);
  padre.dibujar( 50 , 50 );
  madre.dibujar( 450 , 50 );
  hijo.dibujar( 250 , 350 );

  fill(255);
  text("padre",50,40);
  text("madre",450,40);
  text("hijo",250,340);
  text("Click con el mouse para obtener nuevos hijos",50,580);
}
void mousePressed(){
  float adnHijo[] = crossOver( adnPadre , adnMadre );
  hijo = new Fractal( 300 , 200 , adnHijo );
}
```