GENETIC ART

"This process of artificial evolution… it allows the user and computer to interactively work together in a new way to produce results that neither could easily produce alone."
Karl Sims, 1994

INTORDUZIONE

La Gentic Art è un tipo di arte digitale in cui l'opera viene creata tramite un algoritmo genetico. Inizialmente viene presentato all'utente un set random di immagini e gli viene chiesto di selezionare l'immagine ritenuta migliore dal punto di vista artistico. Dall'immagine selezionata viene creato un nuovo set di immagini, dove questa volta ogni immagine è figlia dell'immagine precedente selezionata. Quando un'immagine è figlia di un'altra il cromosoma della prima è simile a quello dell'immagine madre. Questo processo di selezione e riproduzione viene iterato, evolvendosi secondo i gusti dell'utente, permettendo di create delle immagini che sarebbero quasi impossibili da trovare con una semplice ricerca random.
Questi tipi di immagini e questo tipo di arte vengono anche definiti come "Evolutionary Art", "Evolutionary Images", "Gene Art", "Algorithmic Art".

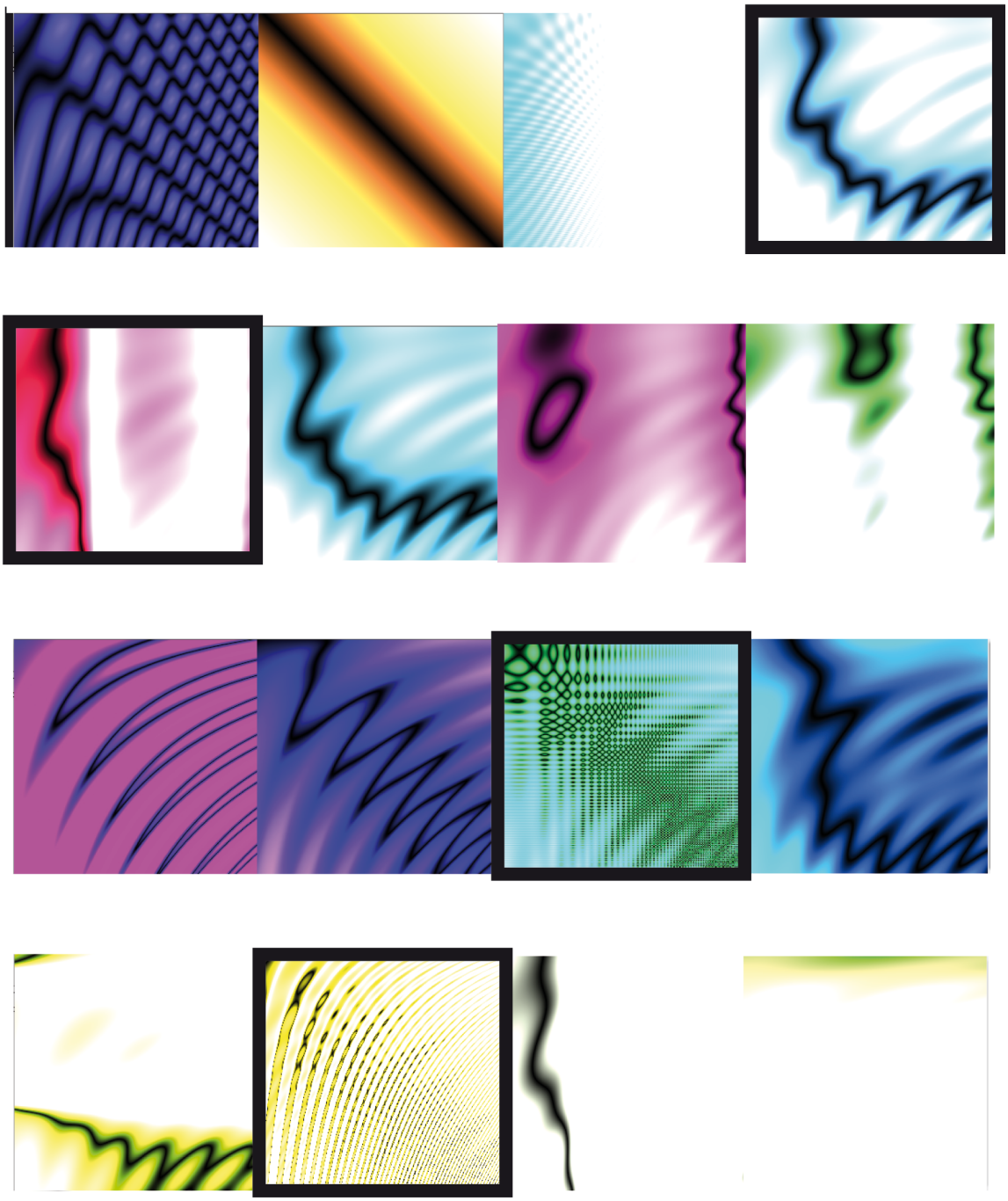
La Genetic Art è un nata negli ultimi decenni. Uno dei pionieri di questo campo è stato Karl Simms, il quale ha sviluppato degli algoritmi per produrre immagini "genetiche" e li ha utilizzati anche per studiare l'evoluzione delle piante ("Artificial Evolution for Computer Graphics", *Computer Graphics*, 25(4), July 1991). L'algoritmo che egli sviluppò creava nuove immagini combinando formule matematiche relativamente semplici e chiedeva all'utente di selezionare le immagini ritenute più piacevoli. Ripetendo questo processo le formule selezionate più volte evolvevano e mutavano, diventando i genitori delle generazioni future.

FUNZIONAMENTO

Un'immagine "genetica" è definita tramite una funzione, utilizzata dal programma per calcolare il colore di ogni pixel dell'immagine.
Ogni funzione è composta da un numero variabile di sottofunzioni. Le sottofunzioni e le operazioni che le uniscono sono inizialmente scelte in modo random da un piccolo database di funzioni e operazioni. Una volta selezionata una certa immagine essa diventa "l'immagine madre". Le funzioni che definiscono le immagini figlie sono calcolate a partire dalla funzione dell'immagine madre: le funzioni figlie ereditano alcune sottofunzioni ed alcune operazioni dalla funzione madre, mentre altre vengono sostituite con delle altre sottofunzioni e operazioni estratte in modo random dal database.

RISULTATI

Nella figura seguente ho riportato 6 iterazioni dell'applicazione. Ad ogni livello è selezionata l'immagine che è stata cliccata per generare le immagini figlie.



Di seguito sono riportate le funzioni da cui sono state generate le immagini riportate sopra. Le funzioni sottolineate sono quelle corrispondenti alle immagini cliccate.

1.
tan(cos(sqrt(x*y*x*y)))+sin(x*x)
y-abs(x)
abs(x*3)-sin(x*x*x-y*y*y)-sin(y+x*y*x+y*y)
y-abs(x)+sin(x)+cos(x)+sin(y)+cos(y)+cos(y)*sin(x*y)

2.
y-abs(x)/sin(x)+cos(x)+sin(y)+cos(y)+cos(y)*sin(x*y)
y-abs(x)+sin(x)+cos(x)+sin(y)+cos(y)+cos(y)*sin(x*y)
y-abs(x)*sin(x)+cos(x)+sin(y)+cos(y)+cos(y)*sin(x*y)
y-abs(x)*cos(x)*sin(x)+cos(y)*sin(y)+cos(y)*sin(x*y)

3.
y-abs(x)*tan(cos(sqrt(x*y*x*y)))+cos(y)*sin(x*y)
y-abs(x)-tan(cos(sqrt(x*y*x*y)))+cos(y)*sin(x*y)
y-abs(x)-cos(y*y*y)+cos(x*x*x)+cos(y)*sin(x*y)
y-abs(x)-cos(y)+sin(y)+cos(x)+sin(x)+cos(y)*sin(x*y)

4.
y-abs(x)/cos(y)+sin(y)+cos(x)+sin(x)+cos(y)*sin(x*y)
y-abs(x)/sin(x*y*x)+cos(y*x*y)+cos(y)*sin(x*y)
y-abs(x)/sin(x)+cos(x)+sin(y)+cos(y)+cos(y)*sin(x*y)
y-abs(x)/x+abs(y)+cos(y)*sin(x*y)

L'ALGORITMO

L'APPLET
Lanciando l'applet GeneticArt.jar si apre una finestra che contiene 6 campi: le quattro immagini, il bottone reset e il bottone exit.

- le immagini: le immagini che vengono visualizzate appena viene aperta l'applicazione sono delle immagini calcolate in modo random. Una volta visualizzate si può cliccare su una delle immagini per creare e quindi visualizzare le quattro immagini figlie prodotte dalla immagine selezionata.
- Reset: cliccando su reset si interrompe il processo genetico in corso e si visualizzano quattro nuove immagini random, da cui si può ripartire con il processo genetico.
- Exit: l'applicazione viene chiusa.

IL CODICE
L'algoritmo che ho sviluppato è scritto in Java e si serve, oltre alle librerie standard, anche del pacchetto "formula"
(http://www.japisoft.com/formula/javadoc/com/japisoft/formula/Formula.html).

L'applicazione si articola in quattro classi: Main, Interfaccia, GenerateImage, Archivio.

Archivio
La classe Archivio rappresenta il piccolo database di funzioni e operazioni. Ho definito 50 funzioni e 4 operazioni (+, -, *, /).
La classe contiene anche due semplici metodi per selezionare in modo random una funzione o un'operazione.

Main
La classe `Main` è una semplice classe che contiene solo il metodo statico main, in cui si può regolare la dimensione delle immagini che vengono prodotte e istanzia un oggetto della classe Interfaccia.

```
public class Main extends Applet{
 public static void main (String[] args){
       int img_dim = 300;
       new Interfaccia(img_dim);
 }
}
```

Interfaccia
La classe Interfaccia ha il compito di amministrare l'interfaccia (basata sulle classi della libreria java.awt).
Tra i suoi campi statici ci sono quattro oggetti di `GenerateImage`, che rappresentano le quattro immagini che vengono visualizzate, e i quattro relativi bottoni, più i due botton di reset ed exit.

```
public class Interfaccia extends Panel implements ActionListener{

       public static GenerateImage img1, img2, img3, img4;
       public static BufferedImage bi1, bi2, bi3, bi4;
       public static int dim;
       public static int scelta = 0;

       private static JFrame frame  = new JFrame("Genetic Art");
       private JButton but1, but2, but3, but4, reset, exit;
```

Il costruttore di Interfaccia inizializza il pannello di interfaccia, istanzia quattro oggetti di `GenerateImage` che rappresentano le quattro immagini e le associa a quattro bottoni. I quattro bottoni rappresentati le quattro immagini e i due bottoni di reset ed exit quindi invocano il metodo `addActionListener` dell'interfaccia `ActionListener`. In questo modo essi diventano sensibili all'evento "button click" che lancia il metodo `actionPerformed`.

```
       public Interfaccia (int img_dim) {

                     frame  = new JFrame("Genetic Art");
                     dim = img_dim;
                     frame.setBounds(0, 0, img_dim*3, img_dim*2);
                     frame.setLayout(new GridLayout(2,3));
                     frame.setVisible(true);

                     img1 = new GenerateImage(dim);
                     img2 = new GenerateImage(dim);
                     img3 = new GenerateImage(dim);
                     img4 = new GenerateImage(dim);

                     System.out.println();

                     //bottoni
                      but1 = new JButton (img1.getImageIcon());
                      but2 = new JButton (img2.getImageIcon());
                      but3 = new JButton (img3.getImageIcon());
                      but4 = new JButton (img4.getImageIcon());
                      reset = new JButton ("RESET");
                      exit = new JButton ("EXIT");
```

```
                //display
                addComponent(frame, but1, 0, 0, 1, 1);
                addComponent(frame, but2, 1, 0, 1, 1);
                addComponent(frame, reset, 2, 0, 1, 1);
                addComponent(frame, but3, 0, 1, 1, 1);
                addComponent(frame, but4, 1, 1, 1, 1);
                addComponent(frame, exit, 2, 1, 1, 1);

                frame.setVisible(true);
                frame.setResizable(true);

                but1.addActionListener(this);
                but2.addActionListener(this);
                but3.addActionListener(this);
                but4.addActionListener(this);
                reset.addActionListener(this);
                exit.addActionListener(this);
        }
```

Il metodo `actionPerformed` dell'interfaccia `ActionListener` modifica il valore del campo statico `scelta` a seconda di quale bottone è stato premuto. Quindi invoca il metodo `run`.

```
        public void actionPerformed(ActionEvent e) {

        while (scelta == 0){
              if(e.getSource() == but1){
                    scelta = 1;
              } else if (e.getSource() == but2){
                    scelta = 2;
              } else if (e.getSource() == but3){
                    scelta = 3;
              }  else if (e.getSource() == but4){
                    scelta = 4;
              } else if (e.getSource() == reset){
                    scelta = 5;
              } else if (e.getSource() == exit){
                    System.exit(0);
              }
              }
              run(scelta);
        }
```

Il metodo run si pulire il pannello dalle immagini precedenti e di sostituirle con nuove immagini (con relativi bottoni).

```
        private void run(int sc){

              if (sc == 5){
                    img1 = new GenerateImage(dim);
                    img2 = new GenerateImage(dim);
              img3 = new GenerateImage(dim);
                    img4 = new GenerateImage(dim);
              }else{
                    String[] parent = AnalisiScelta(scelta);
                    img1 = new GenerateImage(parent, dim);
                    img2 = new GenerateImage(parent, dim);
                    img3 = new GenerateImage(parent, dim);
                    img4 = new GenerateImage(parent, dim);
              }
```

```
        frame.remove(but1);
        frame.remove(but2);
        frame.remove(but3);
        frame.remove(but4);

        //bottoni
         but1 = new JButton (img1.getImageIcon());
         but2 = new JButton (img2.getImageIcon());
         but3 = new JButton (img3.getImageIcon());
         but4 = new JButton (img4.getImageIcon());

         //display
        addComponent(frame, but1, 0, 0, 1, 1);
        addComponent(frame, but2, 1, 0, 1, 1);
        addComponent(frame, reset, 2, 0, 1, 1);
        addComponent(frame, but3, 0, 1, 1, 1);
        addComponent(frame, but4, 1, 1, 1, 1);
        addComponent(frame, exit, 2, 1, 1, 1);

        frame.setVisible(true);
        frame.setResizable(true);

        but1.addActionListener(this);
        but2.addActionListener(this);
        but3.addActionListener(this);
        but4.addActionListener(this);

        scelta = 0;
    }
```

GenerateImage
La classe GenerateImage, infine, è la classe cuore della computazione genetica. Essa gestisce le funzioni e crea da esse le immagini, che poi vengono passate a Interfaccia.

```
public class GenerateImage extends Applet implements ImageProducer{
```

Essa istanzia un oggetto dell'Archivio per avere accesso alle funzioni e alle operazioni.

```
    public static Archivio A = new Archivio();
```

Quindi per oggi immagine si definisce un array di stringe che contiene le sottofunzioni e le operazioni che poi vanno a comporre la formula che genera l'immagine.

```
    public static int dim;
    public Formula tot;
    private static String form = "";
    public BufferedImage image;
    private static String[] funz = new String[7];
    private int num_op = 0;
    private static int funz_lung;
```

Per questa classe esistono due costruttori: il primo viene invocato alla prima iterazione o quando viene dato il comando di reset e crea una funzione random; il secondo invece prende in input anche la funzione dell'immagine madre in modo da generare da questa le funzioni dell'immagine figlia.

```
//costruttore libero
public GenerateImage(int d){
	dim = d;
	image = new BufferedImage(dim, dim, BufferedImage.TYPE_INT_RGB);

	CreateFunz();
	tot = new Formula (form);

	while(!CalcImage()){
		CreateFunz();
	}
}

//costruttore con funzione definita
public GenerateImage(String[] f, int d){
	dim = d;
	image = new BufferedImage(dim, dim, BufferedImage.TYPE_INT_RGB);

	String form = SviluppaFunz(f);
	tot = new Formula (form);

	while(!CalcImage()){
		System.out.println("ERRORE: ricalcolo");
		CreateFunz();
	}
}
```

I metodi che calcolano la formula di un immagine sono due: `CreateFunz` se l'immagine non ha una madre e `SviluppaFunz` se invece l'immagine è generata da un'immagine di partenza.
Il metodo `CreateFunz` crea la funzione dell'immagine estraendo dall'archivio sottofunzioni e operazioni in modo random. Il numero di sottofunzioni (anch'esso sorteggiato in modo random) che comporranno la funzione finale è compreso tra un minimo di uno a un massimo di cinque.

```
private void CreateFunz(){
	num_op = (int)(Math.random()*3);

	int cont = 1;
	funz[0] = A.getRandFunz();
	form = funz[0];
	for (int i=0; i<num_op; i++){

		funz[cont++] = A.getRandOp();
		funz[cont++] = A.getRandFunz();
		form = form + funz[cont-2] + funz[cont-1];
	}


	funz_lung = cont;
	for (int i = cont; i<funz.length; i++){
		funz[i] = "";
	}
}
```

Il metodo `SviluppaFunz`, invece, crea la funzione dell'immagine partendo dalla funzione dell'immagine madre. Tra i sette campi dell'array `funz` (che comprendono 3 operazioni e 4 sottofunzioni, di cui alcuni campi possono essere vuoti) si sorteggia una sottofunzione o

un'operazione che viene sostituita con una nuova sottofunzione (o una nuova operazione) estratta in modo random dall'archivio.

```
private String SviluppaFunz(String[] funzParent){

      int num_max_fun = (int)( (funz_lung - 1) /2);
      int num_max_op = num_max_fun - 1;

       int quale_funz = (int) (Math.random()*num_max_fun) *2;
       int quale_op = ((int) (Math.random()*num_max_op) *2 ) +1;

       funz = funzParent;

       if (quale_funz >= quale_op){
             funz[quale_funz] = A.getRandFunz();
       }else{
             funz[quale_op] =  A.getRandOp();
       }
}
```

Infine il metodo `CalcImage` calcola l'immagine finale partendo dalla funzione calcolata da uno dei due metodi precedenti.

```
private boolean CalcImage() {

      boolean ret = true;
      int x = 0;  int y = 0;
      double val_double;
      Color c;
      double i = 0; double j = 0;

      try {

            int rnd_red = (int)(Math.random() * 255);
            int rnd_green = (int)(Math.random() * 255);
            int rnd_blue = (int)(Math.random() * 255);

            for ( i=1; x<dim; i += 0.02){
                  y=0;
                for ( j=1; y<dim ; j += 0.02){

                  tot.setSymbolValue("x", i);
                  tot.setSymbolValue("y", j);

                  val_double =
                  Math.abs(Double.parseDouble(tot.evaluate().toString()));

                  int red       = ((int) (val_double * rnd_red) < 256) ?
                                        (int) (val_double * rnd_red): 255;
                  int green = ((int) (val_double * rnd_green) < 256) ?
                                    (int) (val_double * rnd_green) :  255;
                  int blue = ((int) (val_double * rnd_blue) < 256) ?
                                    (int) (val_double * rnd_blue) :  255;

                  c = new Color (red, green, blue);
                  image.setRGB(x, y, c.getRGB());

                   y++;
                }
```

```
                x++;
            }


        } catch (Exception e) {
            ret = false;
        }finally{
            return ret;
        }

}
```