Practical Assignment for Evolutionary Computation

J.I. van Hemert and A.E. Eiben
jvhemert/gusz@cs.leidenuniv.nl
room 203a/160

April 6, 1999

1 Introduction

In this assignment you are required to build a system that produces images that resemble art by the Dutch painter Piet Mondriaan. The system is based on techniques from the field of Evolutionary Computation.

The main idea is to utilize the users implicitely present concept of Mondriaan art as a selection criterion in an evolutionary algorithm to direct the search. To this end, a window on the screen presents a population of nine images to the user. The user then rates these images, giving higher rates (fitness values) to those that match his/her idea of Mondriaan art closest. After pressing the `Next` button the program generates a new population (a set of images) using an evolutionary algorithm (that is, selection, recombination and/or mutation). This completes one cycle of the system. The process continues by repeating this cycle until the user decides to terminate.

2 The assignment

Implementing a system as described in the introduction is quite a lot of work. Therefore you are provided with a framework with some missing parts and your task is to design and implement the missing components. These components are discussed further on. First we describe the main parts of the framework.

2.1 The framework

To make life somewhat easier a framework is provided that almost provides a complete program. It falls apart into two main parts. The first one is the evolutionary algorithm, which is coded in `C++` using the *Standard Template*

Library[1]. It is completely set up for the task at hand but misses some parts you must fill in yourself. The second part is the user interface which is coded in plain `C` using the *Gtk library*[2] together with a GUI-builder called *Glade*[3]. It is situated in a subdirectory (`mondrian/src/ui`) of the project and needs no alterations of your hand. The whole project will be done on Linux version 2.0.0 or up.

When the program starts it first makes an object called `Mondriaan` which gets a configuration as a parameter in an object called `Configuration`. The `Mondriaan` object holds everything that belongs to the evolutionary algorithm. During the construction of this object the evolutionary algorithm is initialized and the first generation of the population is constructed. All the individuals from this population are immediately asked to draw an image. After this the user interface starts, it draws these images and starts waiting for user input.

When the user presses the `Next` button the evolutionary algorithm is requested to go to the next generation. This means the algorithm reads in the fitness values as set by the user, performs parent selection and genetic operators and creates a new generation. This new generation is decoded, that is, used to draw images again, thereby completing one cycle of the system.

All other parts of the user interface should be intuitively clear. Every action besides the next-function are hidden away into the subdirectory containing all code for the user interface. The only interactions between the user interface and the evolutionary algorithm are the `Next` operation, the reading of fitness values and the drawing of images. But these interactions are already coded and they require no work from you.

2.2 Your work

2.2.1 Finishing the program

The framework misses some important items: a representation (the genotype), genetic operators (recombination and mutation) and an initialization method. It is your job to design a representation which enables you to draw images, hopefully images that resemble Mondriaan art. This representation should be placed in the `Data.h` file. Note that together with your representation you have to come up with a decoder. Every individual is required to draw an images at some point, which means it calls the function `Draw()` of its data with a canvas as parameter. Your decoder (placed in this function) can draw an image on the canvas using three functions provided by the canvas itself. These are:

- `DrawRectange(LeftTopCorner, RightBottomCorner, Color)`
- `DrawHorizontalLine(LeftEnd, RightEnd, Color)`
- `DrawVerticalLine(TopEnd, BottomEnd, Color)`

[1] Information and reference guide at `http://www.sgi.com/Technology/STL/`
[2] Information at `http://www.gtk.org`
[3] Information at `http://glade.pn.org/`

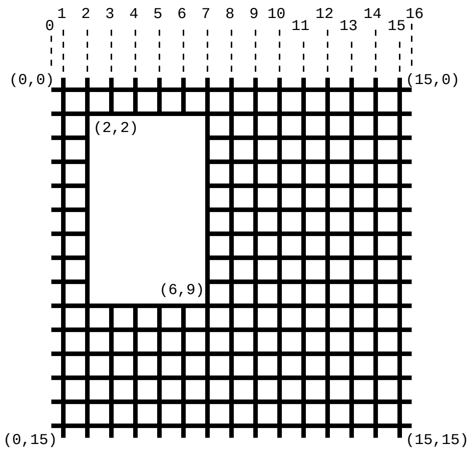

Figure 1: An image drawn using the functions of the canvas. All possible lines are drawn, creating a grid. After that a rectangle is drawn using $(2, 2)$ to $(6, 9)$ as corners.

The coordinates of the squares are all between $(0, 0)$ and $(15, 15)$, and the for the lines between $(0, 0)$ and $(16, 16)$. This to make the search space somewhat smaller. When you draw a rectangle, it will always be drawn between lines. Lines are always drawn on a grid. An example of what an image could look like is presented in Figure 1.

Your representation (the data part) will be subject to genetic operators. The way in which these operators work depend highly on your choice of representation. For instance, if you use a tree representation a crossover for a linear representation is not applicable. Design a good mutation and crossover operator that suits your representation and preferably even facilitates the search for better solutions. You should put the code for these operators in the files `Mutation.h` and `Crossover.h` respectively.

Not only the genetic operators depend on your representation, but also the initialization of the first individuals. Construct an initialization function in the file `Initialization.h` that randomly builds individuals in your representation.

2.2.2 Writing a report

You are requested to write a report in LaTeX that explains your representation, operators and initialization method in detail[4]. It should be clear from this report how your implementation works. Think of it as follows: it should possible to re-implement your work from your report alone. Please do not forget to put your names, numbers and email addresses in this report. We advise you to use the basic report file `report.tex`.

[4] We can provide assistence with Latex on request, furthermore a well-formatted empty document (`report.tex`) in the `mondrian/doc` directory is at your disposal.

3 Mondriaan art

An important question is "How does mondrian art look like?". Of course Mondriaan has produced a lot of different work, but we are only interested in his later work where he limited his painting to straight black lines and rectangles in the colors white, black, red, yellow, and blue. We have put together some rules an images has to follow before we even want to consider it as a Mondriaan:

1. A line always ends (begins) at the edge of the canvas or at a line that runs perpendicular.

2. A rectangle can have one of these colors: red, yellow, blue, black, white.

3. Colored rectangles are completely surrounded by black lines.

4. Lines are always straight (perpendicular to the edges of the canvas).

There are more features that make an image look as if it was painted by Mondriaan, most of these are subjectively chosen by the user. If your system works correctly it should give the user the possibility to explore the large solution space of possible images such that he or she actually ends up with an image that he or she labels Mondriaan like.

Here are some links on the Internet to help you figure out what Mondriaan art looks like:

- Piet Mondriaan - Prints and Posters - Global Gallery
  `http://www.globalgallery.net/bios/mondrian.html`

- Piet Mondriaan Resources and Links
  `http://www.aukword.demon.co.uk/piran/mondrian/resources.html`

4 The material

You can get the files you need from `http://www.wi.leidenuniv.nl/~jvhemert/mondrian`. News and newer versions will also be put here. Please check this page regularly.

5 Delivering your work

Performing this assignment is obligatory if you want to have a mark for the course Evolutionary Algorithms. You are allowed to work on your own, or together with one more person.

5.1 Deliverables

You have to submit the following items in a tar-file which has been compressed using gzip or bzip2. When mailing please use uuencode or mime.

- The source code of your program in the `mondrian/src` directory.
- The report of your paper in LaTeX in the `mondrian/doc` directory called `report.tex`. If you use (PostScript) images, you need to send them too!

This file should be named `mondrian-loginname.tar.gz` (or `.bz2` in case of bzip2) where `loginname` should be replaced by the login name of one of you. The easiest way to produce this file is to run 'make dist' from the `mondrian` directory.

5.2 Deadline

Both the program and the report should compile on the Linux platform as installed in rooms 301 and 307 without errors, otherwise you will not receive a mark. Send in your work before or on **May 11th, 1999** to this address: `jvhemert@cs.leidenuniv.nl` If there are problems please contact Jano van Hemert (default) or Guszti Eiben as soon as possible, either by email or by dropping by.

5.3 Evaluation

Your work will be graded based on the quality of your report (clarity and completeness), the quality of your programming (correctness and readability), and on a test held during the last lectures on May the 12th. Here we will test each program by letting other people try out your final work. Marks for this assignment are weighted by a factor 1/3 in your final mark for the course Evolutionary Algorithms.

Good luck,

Jano van Hemert and Guszti Eiben