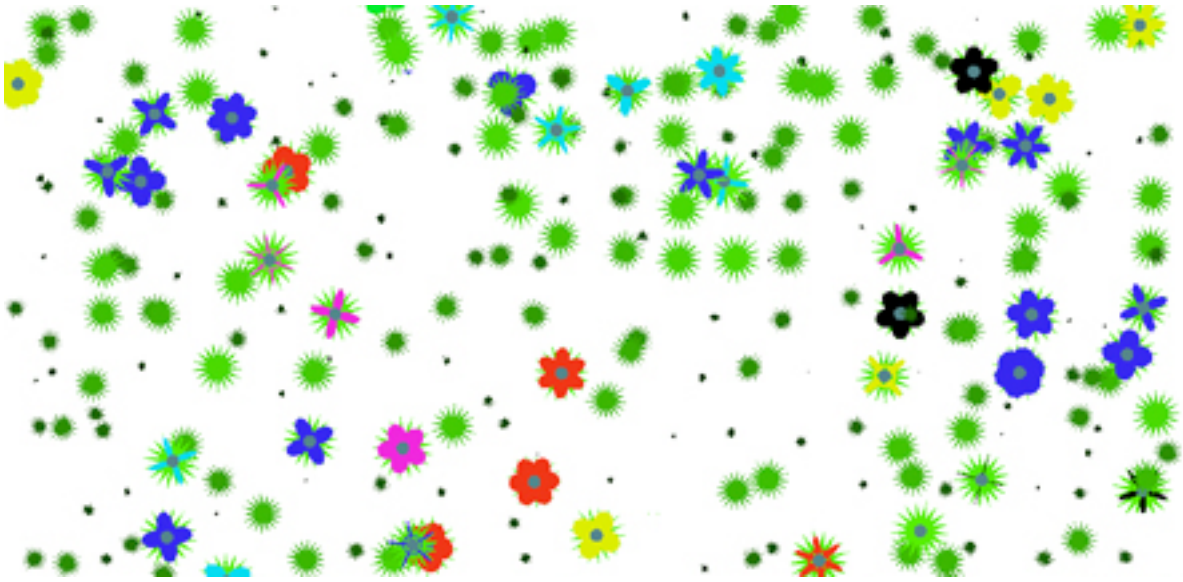
FIT3094 AI, A-Life and Virtual Environments
Assignment 2 - The Artificial, Aesthetic Evolution of Flowers

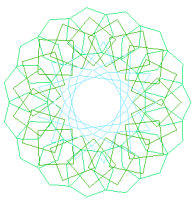
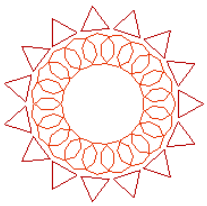
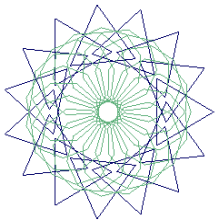
In this assignment you will be guided step-by-step through the design and implementation of an evolutionary algorithm operated by aesthetic selections made by a human user. The subject of the evolutionary process will be the arrangement, shape, colour and structure of a radially symmetrical geometric form: a flower or snowflake-like pattern.

Before you begin this exercise, you must have prepared by reading through the iBook, *BIOLOGICAL BITS, a brief guide to the ideas and artefacts of computational artificial life, Alan Dorin, Animaland, 2014*, section 7.2 pp. 121-131. The assignment builds upon this background knowledge.



Overview

The floral shape that your software will evolve will be generated by rotating a series of *petals* around concentric rings of different diameter making up its *corona*. The diameters of the coronal rings will be specified within a flower's *genotype*, its genes, as will the selection of petal shape, size, colour and number.

A human user will select two flowers from a set displayed in a grid on the screen. These flowers that are subject to human assessment are the *phenotypes* created from the genotypes stored in the software. The selected phenotypes will be used as parents, their genotypes will be subject to crossover and mutation and a new population of offspring will be generated for display. As this process is repeated multiple times, human users will be able to breed floral patterns to their personal taste.

This exercise will give you hands-on experience in coding a simple evolutionary algorithm and allow you to experiment first-hand with evolution's power to generate the diversity typical of life on Earth.

You must submit your code in C++ and a PDF document responding to the questions and tasks set in Parts 4 & 5 of this specification. **During the designated lab session you must demonstrate** your code performing the tasks set in Part 4 of this specification. You must explain how your code works and answer the questions of your lab demonstrator about its operation.

Without a successful demonstration you will receive a mark of 0-N for this assignment.

Part 1 : Drawing individual petals

You may use the sample OpenGL code provided online in files GL_routines.cpp, GL_routines.h, main.cpp as the skeleton of your assignment. This code sets up the window and OpenGL environment for you.

1) Design and implement a class *Flower* that:

a) To begin with, will be used to store parameters for one petal. The class' parameters include: radius, red, green and blue colour components (floating point values (0 to 1.0)) and the petal's number of edges. Store the parameters in a double-type array arranged like this:

```
[petalRadius, red, green, blue, numberOfEdges]
```

This is an ugly way to store data but it will be used later by the evolutionary algorithm.[1]

Include a standard text-based output method to print the Flower's parameters to the console for use in debugging and analysis.

Convert the *drawPetal()* method provided in Appendix 1 to a member of the Flower class.

[5%]

b) Include a call from the Flower class' constructor to a new class method *initialiseRandomly()* that should generate *valid* random values for all a Flower's parameters by calling helper functions to generate values of the correct type and range.

[5%]

c) Be sure to seed the random number generator *once* (and once only) at the start of program execution with a known seed. Then, when you need to re-run the program to debug or to revisit an earlier evolutionary sequence you can do so by specifying the same random number seed as used previously.

[5%]

2) Design and implement a class *World* that:

a) Has a container for a set of N^2 Flowers where N is a runtime user-specified positive integer between 1 and 12.

[5%]

b) Has a method to iterate through the container and draw each Flower at a location centred on an NxN grid. Appendix 2 provides sample code to translate a flower to a location on screen.

[5%]

c) Contains data-members, parent1 and parent2 holding indexes into the container in the range of 0 to (N^2 -1) which identify two currently selected parent Flowers. Be sure to check the values of these variables if the user adjusts the value of N at runtime.

d) Contains methods to set valid numerical values for parent1 and parent2 by entering them from the keyboard.

e) Highlights which Flowers are currently selected as parents by drawing a wireframe square around them in the grid.

[5%]

Write some test-harness/methods to test all of your code before you move on!

---

[1] A neater alternative is to use sensible built in or complex datatypes to store these values. When the Flower needs to participate in the crossover and mutation operations required for the evolutionary algorithm, these values can be copied into an array. Once the breeding operations are complete and a new offspring Flower has been created, its array values can then be read into its sensibly named and typed data members.

Part 2: Setting up Flower evolution

3) Write a Flower method *crossover()* that crosses the genotype of a calling Flower with one passed as a parameter. Use 1-point crossover.

[10%]

4) Write a Flower method *mutate()* that examines each allele (i.e. each single gene value) in the calling Flower's genotype and decides whether or not to mutate it. Using a class-wide shared data member called *mutationRate* that specifies the probability of a mutation occurring. Set the mutation rate to 0.15. Use this parameter to ensure that on average 15% of the genes in a genotype should be mutated.

If a gene *is* selected for mutation, generate a completely new value for it. Ensure validity of the newly generated value: negative colours, edge steps, or Petal diameters wouldn't be permissible for instance, nor would colour values greater than 1.0. Use the same helper methods called by *initialiseRandomly()* for this purpose.

[10%]

5) Execute your work by writing an evolution loop that begins by generating a set of N^2 Flowers with random traits. The loop should repeatedly ask the user to enter values for parent1 and parent2 showing the currently selected parent Flowers with rectangles around them.

Once the user is satisfied with their parent selections, allow them to hit the "m" key to mate the selected parent Flowers. The parent Flowers will then have their genotypes recombined to make a new offspring Flower. The resulting genotypes may be mutated. The recombination and possible mutation of the parents must occur repeatedly (i.e., N x N times) to generate a new population of offspring Flowers.

[10%]

6) Allow the user to hit the "r" key to completely randomise the population of displayed flowers if they are not satisfied with what they see on-screen.

[5%]

Part 3: Flowers with many petals

In this section you construct complex flowers from many petals arranged in concentric rings.

7) Extend your Flower class by adding two new values to the genotype like this:

```
[petalRadius, red,green,blue, numberOfEdges, ringDiameter, numberOfPetals]
```

The **ringDiameter** specifies the diameter of a circle around which petals of the type specified by the first 5 values will be arranged. The **numberOfPetals** parameter refers to the number of petals arranged equally around the circumference of the ring.

A method for drawing this new flower arrangement is given in Appendix 3. Add it to your Flower class's draw method and use it to start rendering a ring of petals for each flower.

[5%]

8) Lengthen your genotype to include 3 rings of petals by tripling the genotype length and using the extra space to replicate the first genotype's structure with new values. You will need to update the Flower class' crossover, mutation, random initialisation and other methods.

[5%]

9) Modify your genotype to allow each petal ring independently to have wireframe petals, filled petals or petals that are both filled *and* have a wireframe outline. In the petal display method use glBegin(GL_POLYGON) or glBegin(GL_LINE_LOOP) for filled or wireframe drawing.

[5%]

Play with your system to evolve some fancy flowers!

Part 4: Demonstration

During your practical session on the day this assignment is due you will need to demonstrate your software to your lab demonstrator. You will need to have completed the exercises in your own time previously and captured screen-grabs of the results to submit as a **PDF document** with your project code. It *doesn't* matter if your live lab demo matches the exact flowers you evolved and will submit in the PDF document.

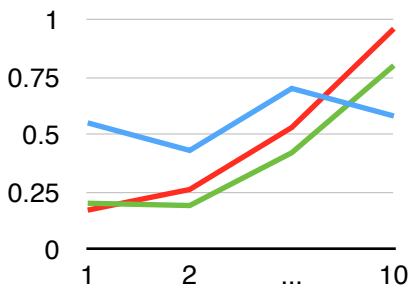
Set up your software with: population = 16 flowers (on a 4 x 4 grid)
mutationRate = 0.15

6) **Demonstrate** your software evolving a *complex*, but to your mind, attractive flower using exactly 5 breeding operations of two parents at each stage from random initialisation to final population.

In your own time, repeat this process and capture images of the pairs of parents in each stage of the evolution of this flower and lay them out in sequence on a diagram clearly showing the progression from generation 0's parents to the final selected 5th generation flower. Beneath the final flower, include a printout of its genotype.

[5%]

7) **Demonstrate** your software evolving an entirely *red* flower, using *up to* 10 breeding operations of two parents at each stage from random initialisation to final flower.



In your own time, add a simple "print" statement to your code to output the mean red, green and blue colour genes of every flower in the current population to the console. Repeat the evolutionary process of the demonstration over exactly 10 generations, always evolving the population towards red. Capture images of the pairs of parents you select in each stage of the evolution of this flower and lay them out in sequence on a diagram clearly showing the progression from generation 0's parents to the final selected flower. Include in your PDF document a single pair of graph axes on which you plot three separate lines: the mean red, green and blue gene values against the breeding generation (see left).

[5%]

8(a) **Demonstrate** the selection of a single flower from the initial population set as both parent1 *and* parent 2. Then mate the flower with itself.

In your own time, repeat this process and capture images of the selected parent and the new population of offspring it generates. Explain in a few sentences what has happened.

[5%]

Part 5: Further exercises for submission (but not demonstration)

8(b) Set your code's mutationRate=0.3. Using the *exact flower* parent used for part 8(a), provide an image of the new population of offspring the parent flower generates. Is the result different to that obtained for part a)? How and why? Try to evolve an entirely red flower whilst always selecting a single flower as both parent1 and parent2. How long does it take? Is it easier or harder than when using two different flowers as parents?

[5%]

9) **BONUS:** Add a new method to your class World that, with a hit of the "f" key, toggles between the *N*x*N* grid layout to a field of flowers consisting of the current population scattered randomly on the screen. Set your population to 144 flowers. In grid mode, evolve an interesting population, then hit "f" and capture a screenshot of the resulting field of flowers.

[+5% bonus. Assignment total capped at 100%]

Appendix 1: drawPetal( ) - requires <math.h> and OpenGL

```
void drawPetal
      (double radius, int numEdges, double red, double green, double blue)
{
     double centX=0, centY=0, centZ=0;
     double currentPointX, currentPointY;
     double angleRadians= 0;
     double stepSize    = (2*M_PI)/(double)numEdges;

     // Set the current colour (white R=G=B=1.0, black R=G=B=0.0)
     glColor3d(red, green, blue);

     // glBegin(GL_POLYGON); can be used in order to draw a filled petal
     glBegin(GL_LINE_LOOP);

     // Traverse around a circle in the specified number of steps
     for (angleRadians=0; angleRadians < (2*M_PI); angleRadians+=stepSize)
     {
      currentPointX = centX+(radius * cos(angleRadians));
      currentPointY = centY+(radius * sin(angleRadians));

      glVertex2f(currentPointX, currentPointY);
     }

     glEnd();
}
```

Appendix 2: translate a petal in the X/Y plane - requires OpenGL

```
glPushMatrix();
glTranslated(Xcoordinate, Ycoordinate, 0);
drawPetal( … );
glPopMatrix();
```

Appendix 3: rotate a series of petals so that they are arranged around a ring and translate them to the circumference of that ring

```
void drawPetalRing
(const double radius, const int numEdges, const double red, const double green,
const double blue, const double ringRadius, const double numberOfPetals)
{
   […  setup code omitted …]

  glPushMatrix();
  for (int i=0; i<numberOfPetals; i++)
  {
     glPushMatrix();

     // move petal to position on the ring circumference (angles in radians)
     glTranslated(ringRadius*cos(i*2.0*M_PI/numberOfPetals),
                         ringRadius*sin(i*2.0*M_PI/numberOfPetals), 0);

     // rotate petal (in degrees) so it points away from the ring's centre
     glRotated((double)i*360.0/(double)numberOfPetals, 0.0, 0.0, 1.0);

     drawPetal( … );

     glPopMatrix();
  }
  glPopMatrix();
}
```