University of London International Programmes

Creative Computing

CO1112 Creative computing I: Image, sound and motion

Coursework assignment 2 2017-18

Aims

The aims of Coursework assignment 2 are:

1. To develop your knowledge and practical experience of programming generative systems, and in particular L-systems.

2. To allow you to explore the variety of forms that can be produced by a simple L-system.

3. To consider how genetic algorithms might be applied to L-systems.

4. To develop your knowledge and practical experience of using object-oriented programming, and in particular how OOP enables you to build up complex scenes in a Processing sketch.

5. To give you the opportunity to combine various techniques you have learned about during the course to produce a creative artefact.

General instructions

There are five parts to this coursework, and you are expected to spend between 20–40 hours on it. Note the marks given for each part, and plan to spend time on each part roughly in proportion to the marks available.

Carefully follow the submission instructions at the end of each part, and the general instructions at the end of the coursework, to ensure that you submit what is required, and that it is in the format required.

Introduction

If you have not done so already, read Sections 6.1 and 6.2 of Volume 2 of the subject guide, on fractals and L-systems. In this coursework we will mostly be using the techniques described in Section 6.2.5.

Download the `Lsystem` sketch code that accompanies this assignment. This is an object-oriented version of the Bracketed L-system code shown in Figure 6.13 in the subject guide, with various extensions. Open the sketch in Processing and study the code[1]. If you are new to object-oriented programming, you may find it useful to review the tutorial on the Processing website at `http://processing.org/tutorials/objects/`.

Notice in particular that the `Tree` class (defined in `Tree.pde`) has a constructor method which performs initialisation of the member variables. The constructor is called automatically when

[1] As a minor point of explanation, you will notice that the names of the member (instance) variables in the Tree class start with an "m_" prefix. This makes it clear which variables are member variables and which are not (e.g. local or global variables). This is one of a number of conventions you might come across for this purpose when reading other peoples object-oriented code (e.g. some programmers simply use "_" rather than "m_").

a new `Tree` object is created (which is accomplished using the new operator, as happens in the `setup()` method in `LSystem.pde`). Also notice that the `Tree` class has a complete set of *setter* methods which allow you to set any of a *Tree* object's member variables independently after the constructor has been called.

Notice also that the `setup()` method in `LSystem.pde` ends with three lines which call setup code specific to Questions 1, 2 and 3 below. When answering these three questions, ensure that the correct line is uncommented, and that the other two lines are commented out.

Run the `Lsystem` sketch and see what it does.

Part 1: L-system basics [15%]

- The Tree constructor defines the 'F' substitution rule of the L-system as the string F[+F]F[-F]F. Explain the meaning of each of the five distinct symbols in this string (i.e. 'F', '[' , ']' , '+', '-'), and then describe (using a diagram if helpful) the overall result of how this string (without any substitutions) would be graphically interpreted if passed in to the turtle() method.

[15%]

What to submit for Part 1
Submit your written answer to this question as part of a single PDF submission file containing your answers to all of the questions in this coursework. Ensure it is clear which section of your submission is answering which question.

Part 2: Generating random rule strings [25%]

Your task in this part is to implement a method to generate strings composed of randomly selected characters from our L-system's allowed alphabet. In Part 3 we will use this method to explore the graphical output of random rules, but for now we will just concentrate on generating the random strings themselves.

Ensure that the line "`question2setup();`" in the setup() method in `LSystem.pde` is uncommented, and that the "`question1setup();`" and "`question3setup();`" lines are commented out.

At the bottom of `Tree.pde` you will see that there is a method `generateRandomRule()` which returns a `String`. However, most of the method's implementation is missing! To complete this Part you need to implement the missing code for this method.

Notice that the `question2setup()` method in `LSystem.pde` calls `generateRandomRule()`30 times, printing the result to the console output each time. If you run the sketch as supplied, you will see that all 30 strings are empty. When you have implemented `generateRandomRule`() correctly, you should see 30 different strings being produced.

Proceed as follows:

- Write Processing code to generate a string of length 3 characters, where each character is randomly selected from the alphabet {'F', 'H','f', 's','+', '-'} (i.e. the code might generate strings such as 'Fsf', '++H' '-+f', etc.). You will need to use Processing's `random()` function to achieve this.

[8%]

- Improve your implementation by allowing the generated string to be of different lengths each time the code is called. Restrict the range of possible lengths to the range 1 to 8 characters inclusive (i.e. the code might generate strings such as ‘F’, ‘++H-ff’ ‘-sFHFfF+’, etc.).

  [4%]

- Consider whether there are any potential problem cases with using the generated strings as rules in the L-system. Fix your code so that it does not generate the problem cases.

  [5%]

- Add the characters ‘[’ and ‘]’ to the alphabet of characters that might be included in the string. Note that you cannot add these two characters completely at random! A ‘]’ *cannot* appear before a ‘[’, and a ‘[’ *must* be followed by a ‘]’ later in the string.

  [8%]

What to submit for Part 2
In your PDF submission file, write a few sentences about which parts of the question you successfully completed. If you had problems, explain what they were, and how you tried to overcome them, even if you were unsuccessful. Include in your PDF submission a print-out of the final version of your `generateRandomRule()` method, and also a sample output of 30 random rules produced by the code. Also submit your revised `Lsystem.pde` and `Tree.pde` files in a directory called `Parts2and3` (see the instructions at the end of the coursework for details, and note that your submitted .pde files should contain your code for both Part 2 and Part 3 – do not submit separate .pde files for these two parts.)

Part 3: Exploring the output of random rules [15%]

In this part we will explore the graphical output of using randomly generated rules, by setting the L-system rules to random strings using the method you developed in Part 2.

Ensure that the line “`question3setup();`” in the `setup()` method in `LSystem.pde` is uncommented, and that the “`question1setup();`” and “`question2setup();`” lines are commented out.

Study the method `question3setup()` in `LSystem.pde`. Notice that it calls the method `generateRandomRule()` to generate a random string, and then calls the tree’s `setFRule()` method so that it uses this string as its F substitution rule.

Note that if you were unable to complete Part 2, you can still do some exploration here by hand-coding some random rules into the `question3setup()` method.

- Improve `question3setup()` so that it saves the output in a file. You will need to use Processing’s `save()` method to do this. Run the sketch at least 12 times to generate 12 different images. You will need to find some way to give each image a different filename to avoid overwriting them. If you can’t manage that, you will need to manually rename the new image file before running the sketch again.

  [15%]

What to submit for Part 3
Include in your PDF submission a print-out of the final version of your `question3setup()` method, followed by the 12 (or more) images that you generated. Also discuss what you did (including any extensions you made beyond what was asked for, if any). Discuss any observations you made about the outputs, including any problems encountered, and your thoughts about the generated patterns. As explained in Part 2, you should also submit your revised `Lsystem.pde` and `Tree.pde` files containing your code for both Part 2 and Part 3, following the submission guidelines at the end of this coursework.

Part 4: Using Genetic Algorithms with L-systems [15%]

If you have not done so already, read Section 6.3 of Volume 2 of the subject guide, on genetic algorithms. An important aspect of a genetic algorithm is the process of *mutation*, which makes a small change in some aspect of a selected parent object in order to generate a slightly different child object.

➢ Write a short essay (approximately 300–400 words) describing the general design of a user-guided genetic algorithm to evolve patterns in our L-system. Concentrate in particular on explaining the operation of a suitable mutation method of the form `String mutate(String rule)` which takes an existing rule string and returns a mutated string. You do not need to implement the method, but just describe how it should work. Also, describe how the user selection aspect of the system might work.
[15%]

What to submit for Part 4
Submit your essay as part of your PDF submission.

Part 5: An extended creative artefact [30%]

➢ Extend the L-system code to produce a creative artefact by incorporating other techniques you have learned about during the *Creative Computing 1* course.

For example, you might choose to incorporate elements such as 3D drawing, animation, texture mapping, colour, or sound. Or you might choose to utilise the object-oriented nature of the sketch to draw various different trees in a single scene. These are just examples, but feel free to use other techniques. Other possible areas to explore include using more sophisticated L-systems, interpreting the L-system state string in alternative ways to that defined in the `turtle` method, or even implementing the kind of genetic algorithm discussed in Part 4.

If you decide to look at more sophisticated L-systems, there are plenty of free sources of further information available online. An excellent example is the book *The Algorithmic Beauty of Plants*, written by Przemyslaw Prusinkiewicz and Aristid Lindenmayer, and available to download at `http://algorithmicbotany.org/papers/#abop`.
[30%]

What to submit for Part 5
In your PDF submission, include a written description of what you have developed. Explain what you tried to achieve and what techniques you have used. Give a general overview of what each section of your code does, and discuss any problems you encountered (and, if applicable, how you overcame them). Include some screenshots of the output of your system. Also include a final paragraph evaluating what you have done: what are its strengths, and

which aspects could be improved upon? In addition to your written submission, also submit the source code for your sketch (and any other necessary files, such as image files, etc.), so that the Examiners can run it. Submit your code for this Part in a directory called `Part5`, following the submission instructions at the end of this coursework.

Submit the following by uploading to the VLE:

1. A single PDF document of your written assignment submission, containing your answers to all the questions and discussions asked for. The file should be named using the following convention: `YourName_SRN_CO1112cw2.pdf`.

2. A zip file called `YourName_SRN_CO1112cw2_Parts2and3.zip`, containing one directory called `Parts2and3`. This directory should contain a single subdirectory called `Lsystem`, and that should contain your two modified `Lsystem.pde` and `Tree.pde` files with your modifications made in Parts 2 and 3 of the coursework.

3. A zip file called `YourName_SRN_CO1112cw2_Part5.zip`, containing one directory called `Part5`, which contains all `.pde` file(s) and any associated data files for the sketch you developed for Part 5. The main `.pde` file should be called `Part5.pde`.

Important notes

Citation and referencing: It is important that your submitted assignment is your own individual work and, for the most part, written in your own words. You must provide appropriate in-text citation for both paraphrase and quotation, with a detailed reference section at the end of your assignment (this should not be included in the word count). Copying, plagiarism and unaccredited and wholesale reproduction of material from books or from any online source is unacceptable, and will be penalised (see: How to avoid plagiarism).

Copying code from other sources: It is a normal aspect of programming to look at other people's code to get inspiration on how to solve a problem. This might extend to directly copying code obtained from elsewhere into your own program. If you do this, you should include a comment in your code to say which part has been copied, and where it came from, acknowledging the original author or source. While looking at and using existing code can be a good way to learn appropriate style, you should ensure you use high quality examples that exhibit good technical ability and programming style.

[END OF COURSEWORK ASSIGNMENT 2]