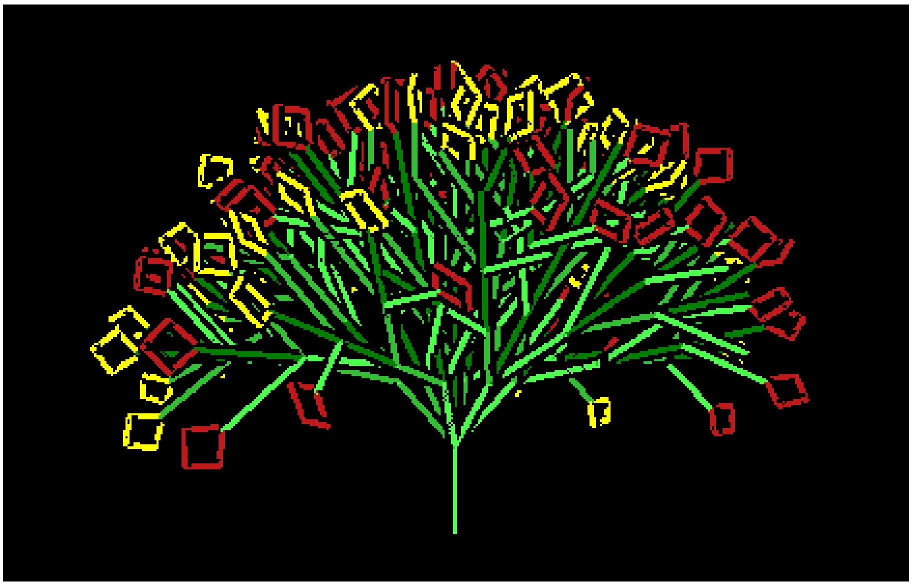
Licenciatura em Engenharia Informática e Multimédia (LEIM)

Modelação e Simulação de Sistemas Naturais

20/21 – Semestre Inverno

Trabalho nº3



LEIM33D-a – 10 janeiro 2021

Luís Fonseca nº 45125

Paulo Jorge nº 45121

Índice

Índice de Figuras

Introdução

Neste trabalho foi feito o estudo do comportamento de agentes autónomos e objetos fractais. Com estes conceitos, foi proposto um trabalho de casa onde era posto em prática, a execução destes dois conceitos, pegando em vários exercícios e gerar diferentes resultados para eventuais estudos e conclusões.

Tarefa A – Função Logística

Alínea 3

Como se pode observar nas figuras apresentadas anteriormente, o comportamento observado para os diferentes valores são:

Decaimento: $0 < r < 1$

Estável: $1 < r < 3$

Periódico: $3 < r < 3.57$

Caos: $r > 3.57$

As várias condições iniciais testadas para cada valor do parâmetro r demonstram que o comportamento apresentado para os valores de r iguais a 3.628, a 3.835, a 3.92 e a 4, demonstram que o sistema possui uma grande sensibilidade aos valores iniciais (efeito borboleta). Desta forma, o sistema não apresenta padrões, sendo por isso considerado um sistema caótico.

Estes sistemas podem ser observados nas imagens abaixo:

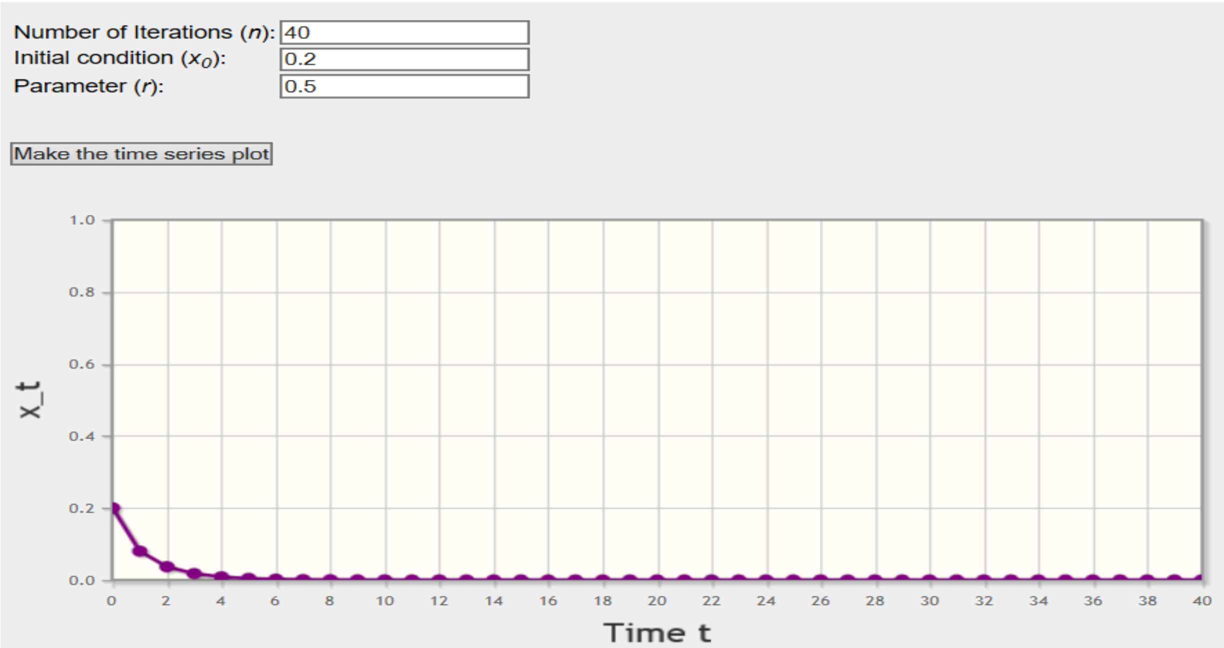


Figura 1 - primeiro ensaio

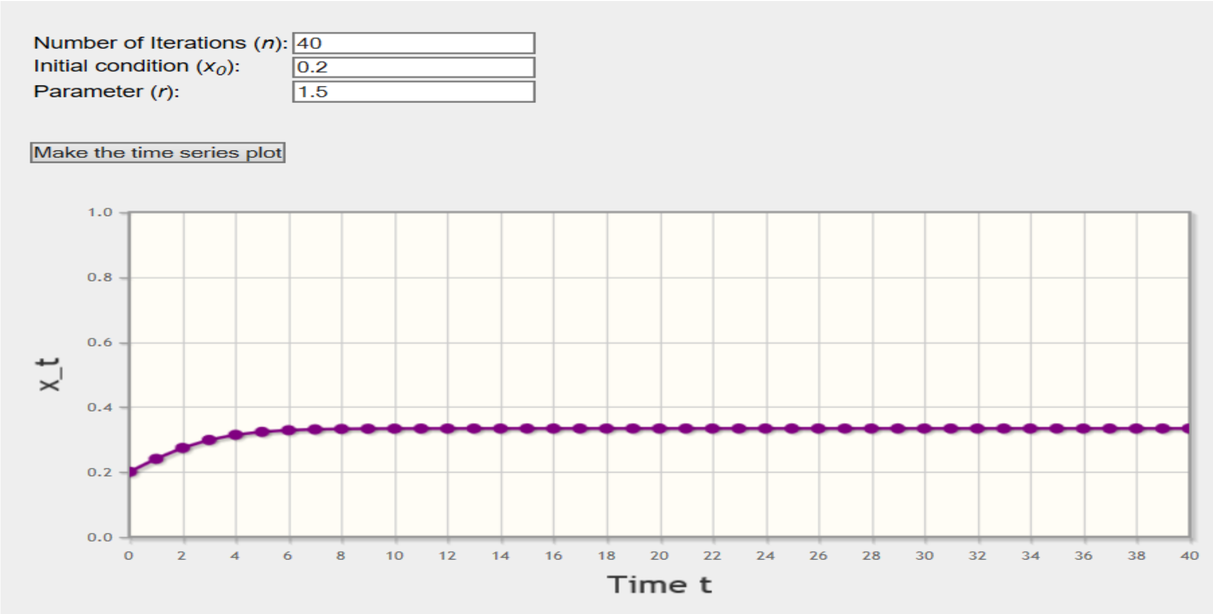


Figura 2 - segundo ensaio

Tarefa B – Jogo do Caos

Foi efetuado o jogo do caos, criando iteração com o utilizador. Para isso, o grupo fez a separação do triangulo e do quadrado gerado pelo método do jogo do Caos. Para isso foi feito com o utilizador pode-se carregar com os botões do rato, alternando entre o desenho do triângulo e do quadrado.

O resultado deste exercício pode ser visualizado em baixo:

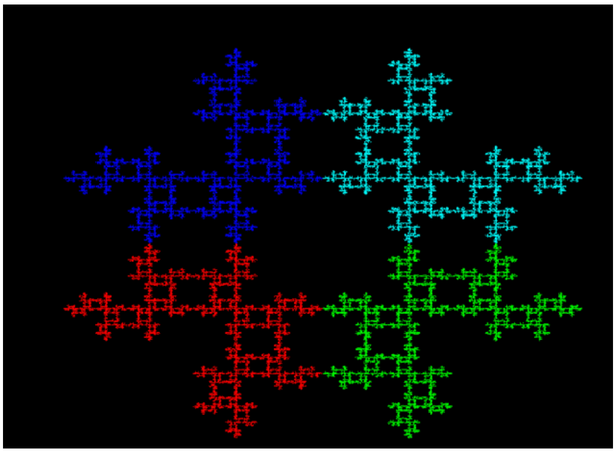


Figura 4 - desenho do quadrado

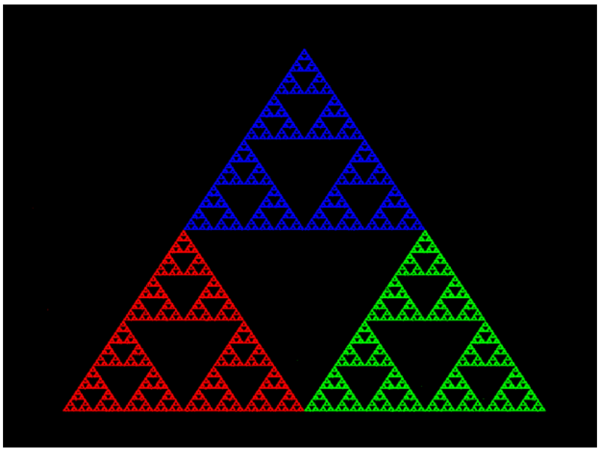


Figura 3- desenho do triangulo

Aplicamos também algumas restrições ao jogo do caos. Para isso recorreu-se à classe *PVector*, esta classe permite criar vetores, e conseguir alterar as suas posições *x* e *y.* Dentro do método *setup(),* é criada uma variável de nome *angle*, que permite calcular o valor do angulo, e alterando entre as diferentes posições. No final do método, são usados os método *mult()* que corresponde a uma multiplicação de u vetor, neste caso, é multiplicado pelo comprimento do nosso canvas, e no final é adicionado, esses valores ao vetor criado.

Foi criado o método *reset()* que faz com que tudo o que é desenhado no canvas seja apagado.

No final, dentro do método *draw()* é onde é desenhado os pontos que permitem a visualização da figura de baixo. Para fazer uma aproximação dos valores, o grupo criou outro Vetor de nome *next*. Dentro deste vetor, é usada a função do processing *floor,* esta função permite encontrar valores mais próximos e igualar a esse valor, neste caso, quando surge um ponto que esteja perto de outro ponto, ele é desenhado, no final são acedidas as posições x e y dos vetores, onde iram ser estar localizadas as posições correntes dos pontos.

Quando corrido o programa, o resultado originado foi este:

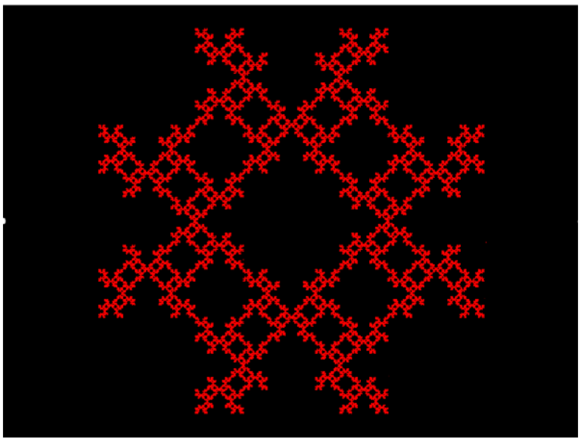


Figura 5 - jogo do caos com restrições

Tarefa C – Gramáticas de Lindenmayer

Neste exercício era pedido que fossem construídos dois fractais, passando uma determinada regra. O método criado foi simplesmente aceder à classe *Regra*, que contém um axioma e uma regra (ambos passados como parâmetro desta classe), e de seguida é colocada regra dentro do array.

A implementação feita para o primeiro fractal, pode ser visto no código abaixo:

De seguida foi corrido o programa, os resultados obtidos foram estes:

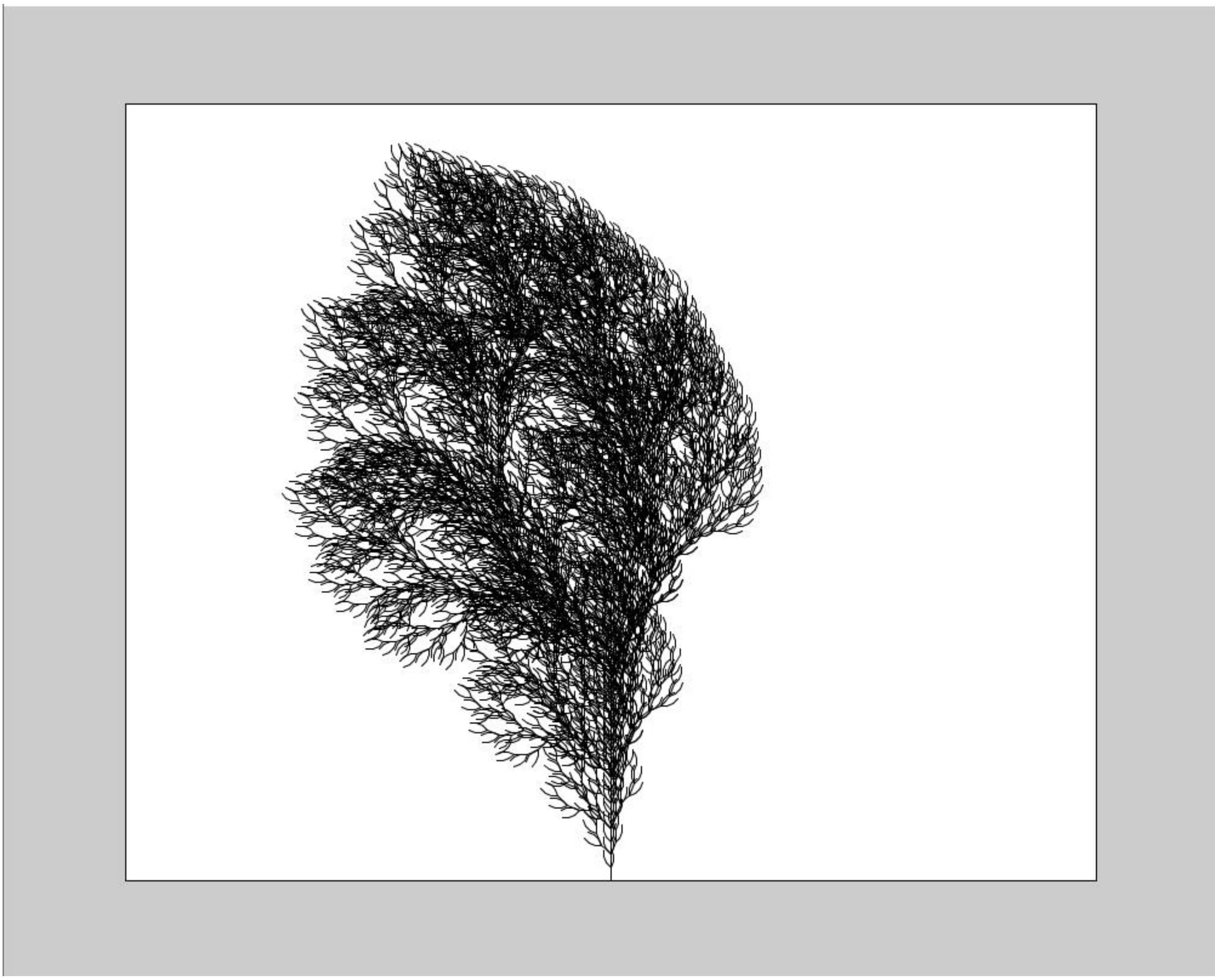


Figura 6 - Fratal gerado com a regra F[+F]F[-F][F]

A implementação do segundo fratal pode ser vista no código abaixo:

De seguida foi corrido o programa, os resultados obtidos foram estes:

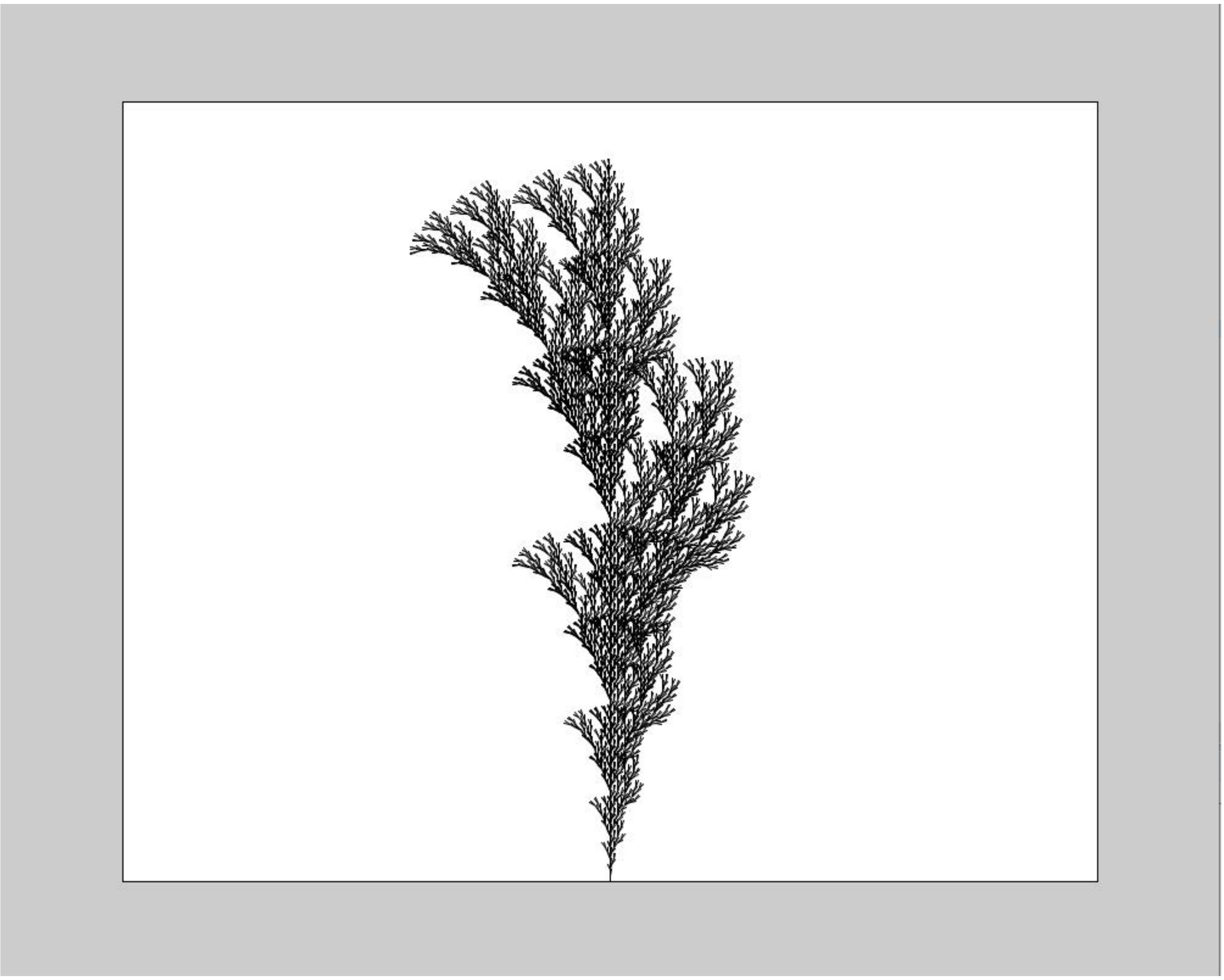


Figura 7 Fratal gerado com a regra F[+F]F[-F][F]

Foi efetuado também outro fractal, mas que desse a criação de uma árvore com frutos, através das seguintes condições:

Variáveis e Constantes:

F, G, +, -, [,]

Axioma: F

Regras:

1. F -> G[+F]-F

2. G -> GG

Para isso, usando o array de nome *RegraSet*, aumentou-se a sua capacidade para duas posições, sendo essas posições, com os respetivos axiomas, no final é chamada a classe *Regra,* para evocar o axioma e aplicar as duas regras escritas.

O resultado pode ser encontrado na figura abaixo:

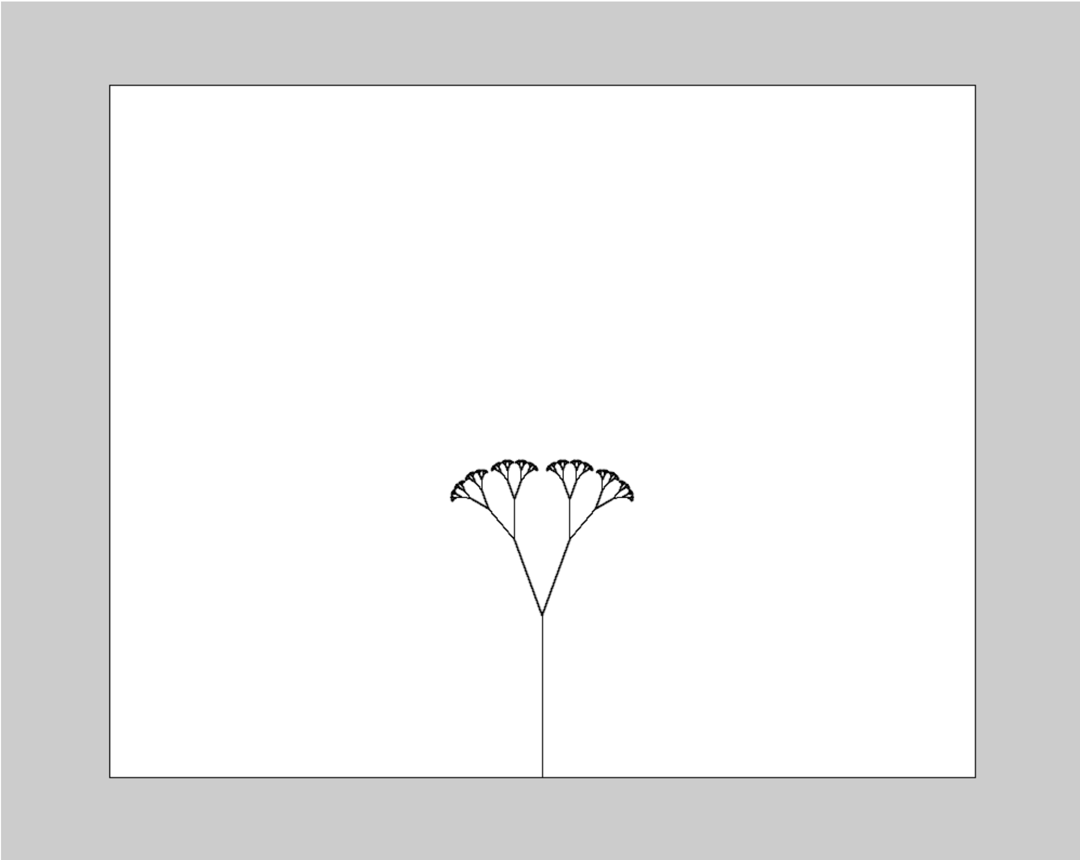


Figura 8 - Fratal criado com as regras G[+F]-F e GG

Efetuamos o desenho da “Curva de Koch”. Para a produção do Koch Snowflake utilizou-se como axioma ***“+F--F—F”*** e como regra ***“F = F+F--F+F “***. A sua dimensão fractal é 1.2644.

O procedimento efetuado para o desenho do fratal é idêntico aos dos exercícios anteriores.

Em baixo, encontra-se uma proposta de resolução:

De seguida foi corrido o programa, os resultados obtidos foram estes:

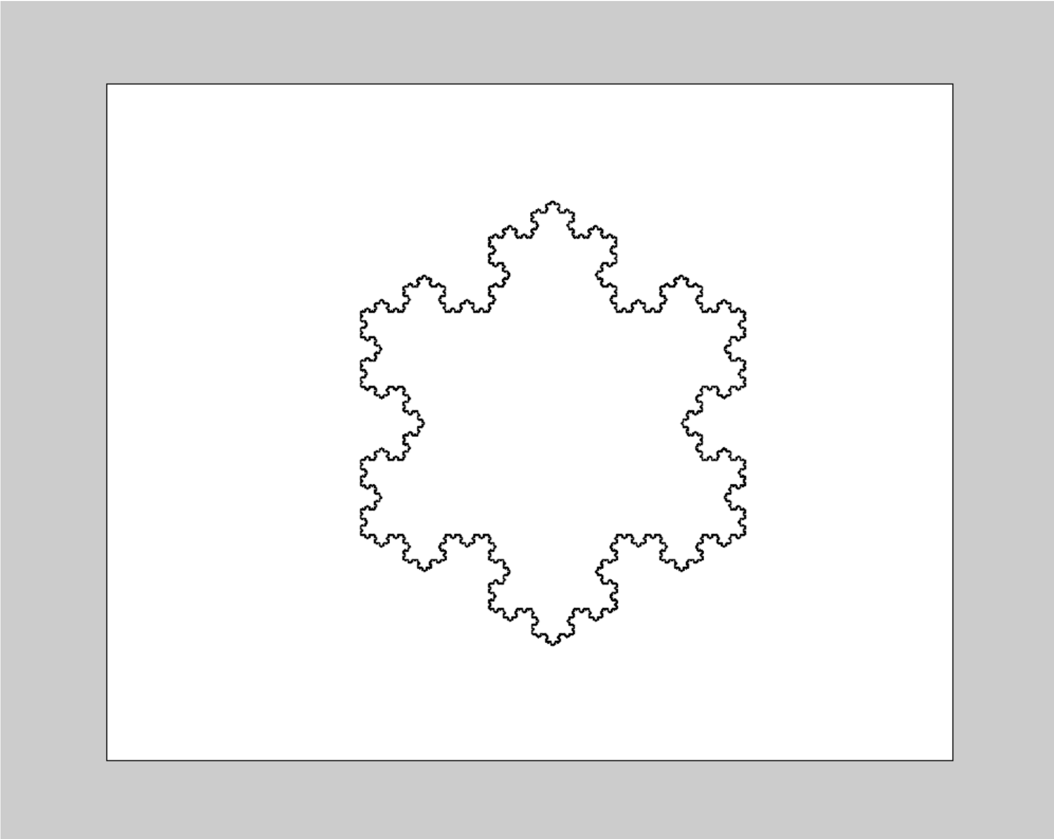


Figura 9 - Koch Snowflake iteração 6

Efetuamos também o desenho de fratais estocásticos, para isso na classe LSystem, criou-se o método Regra50, em que são passadas duas Regras como argumento e devolve um array de Regras.

A cada iteração a regra é atualizada consoante a Regra que calhar no método. Ao juntarmos duas regras de árvores diferentes obtivemos este resultado.

De seguida foi corrido o programa, os resultados obtidos foram estes:

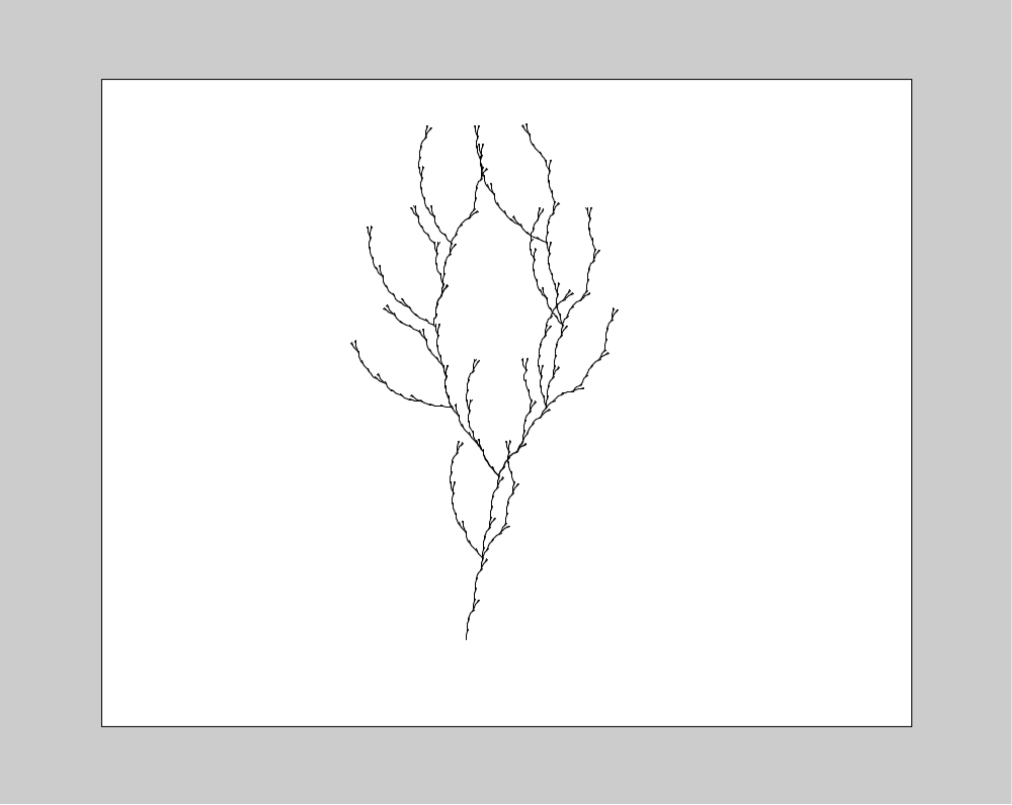


Figura 10 - Árvore com regras estocásticas

Neste exercício era pedido que fosse construído vários fractais, nomeadamente um conjunto de espécies de árvores iguais ou diferentes, neste caso o grupo optou por construir diferentes espécies de árvores.

Para isso foram criados vários axiomas, cada fatal com a sua respetiva regra.

No método *setup()* foram inicializadas todas as arvores que foram construídas, como se pode ver no código abaixo:

Para isso acontecer pensámos em guardar o número da iteração numa variável, e caso a iteração fosse a número x, a árvore daria frutos. Contudo não o conseguimos reproduzir. Uma ideia que também tivemos era a de à medida que a árvore cresce, outras árvores começavam a nascer do solo, como que um fast forward do tempo.

De seguida foi corrido o programa, os resultados obtidos foram estes:

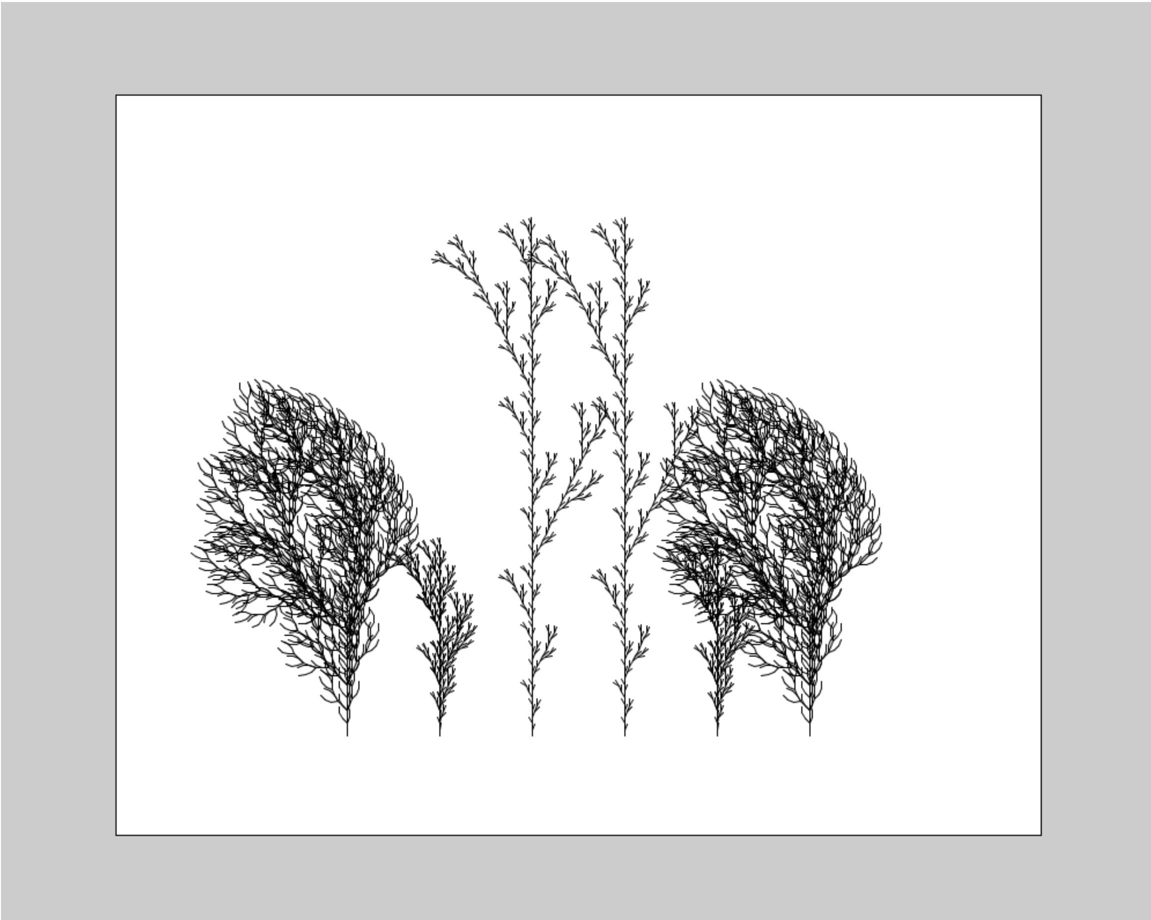


Figura 11 - construção das diferentes árvores com as diferentes regras

Tarefa D – Conjuntos de Julia e Mandelbrot

Para esta tarefa foi pedido a realização dos Conjuntos de Julia e Mandelbrot. Como passo inicial, foram criadas variáveis que correspondem ao comprimento e a largura do canvas. Foram usadas as funções *noLoop()* e *loadPixels()*. O método *noLoop()* permite interrmomper o processo de execução do método *draw()*, e ao usar este método, não é possível aceder aos método *mousePressed* e *keyPressed()*,e o método *loadPixels()* que permite carreger os dados da janela de exibição atual da matriz pixels[].De seguida é calculado o valor máximo de cada um dos eixos, de seguida é feito o incremento de x e y para cada pixel. A seguir é começado o valor de x e y, é de notar que para este exercício, o grupo considerou o modelo de uma expressão de um complexo da forma z = a+bi, sendo a o valor de x e b o valor de y. No final, é desenhado os pixéis com uma cor, neste caso, o grupo obtou por colorar a cor de preto.

A visualização deste código pode ser encontrada no relatório na secção “Anexos”:

Quando corrido o programa, o resultado originado foi este:

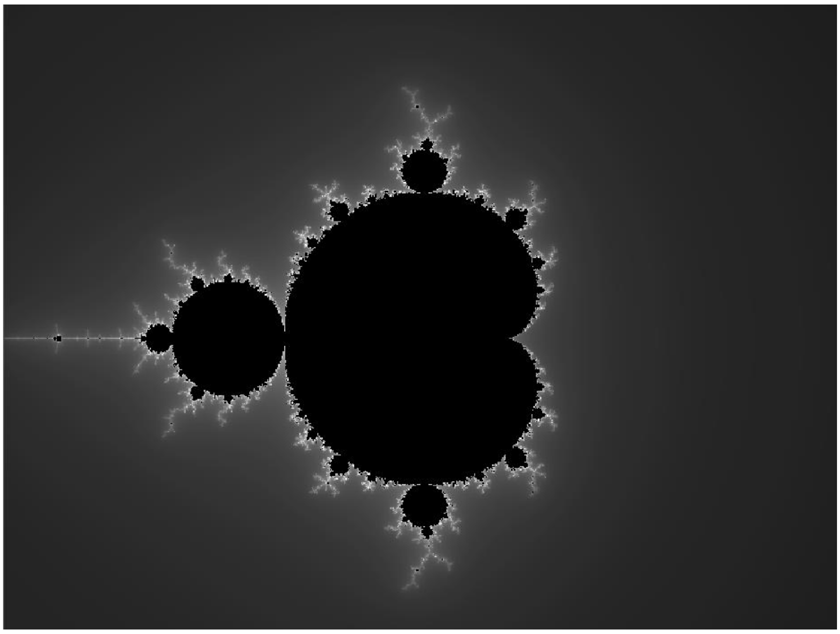


Figura 12 - conjunto de Julia e Mandelbrot

Tarefa E – Ferramentas para criar fractais

Para efetuar o estudo acerca dos diferentes tipos de fractais, o grupo descarregou uma aplicação, feita em Java de nome *FractalGrower.Jar*, esta aplicação permite fazer o desenho de vários fractais, dando um determinado ângulo, e rodando para um dos lados, em baixo são apresentados vários exemplos, e uma breve explicação acerca do "Fractal Grower".

Sobre Fractal Grower:

O fractal Grower permite gerar fractais e sistemas Lindenmayer com muita facilidade.

O primeiro fractal que nos aparece, é o chamado "Paper Folding fractal" que tem como base ir dobrando um pedaço de papel de tamanho infinito, em ângulos específicos e verificar qual o resultado, escolhendo um ângulo qualquer entre 0 e 180 depressa podemos ver resultados interessantes. Os resultados podem ser encontrados nas figuras abaixo:

- A primeira imagem corresponde ao ângulo de 1º
- A segunda imagem corresponde ao ângulo de 45º
- A terceira imagem corresponde ao ângulo de 90º
- A quarta imagem corresponde ao ângulo de 130º

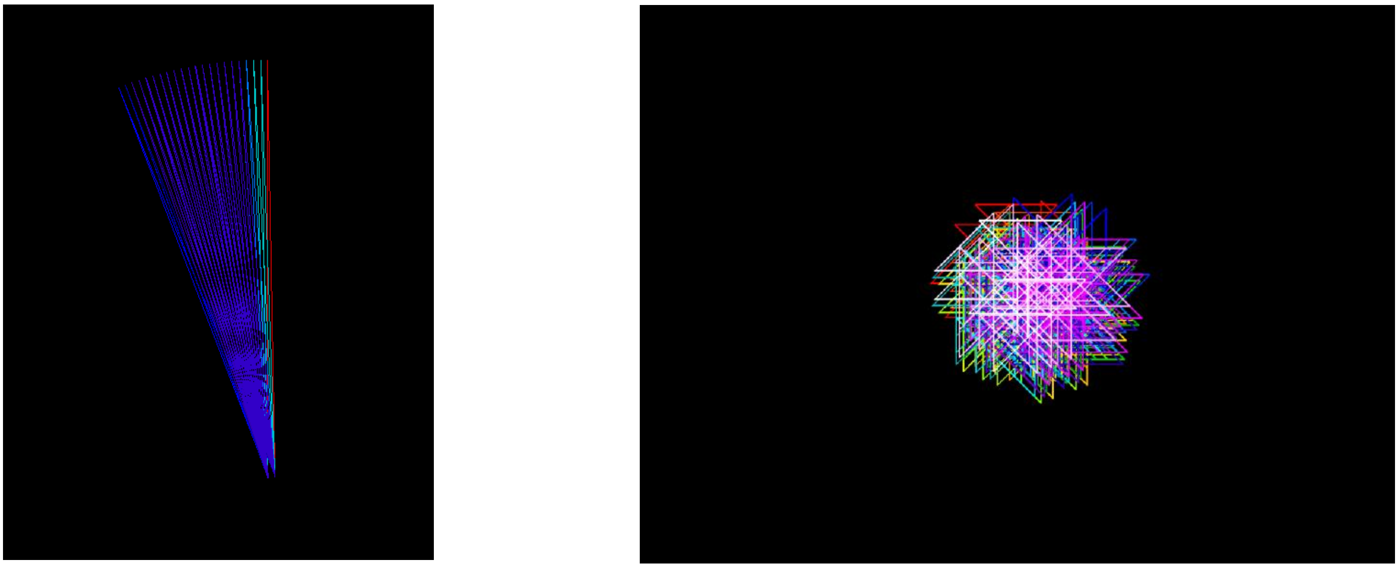


Figura 13 - Fratal Paper Folding com angulos de 1º e 45º

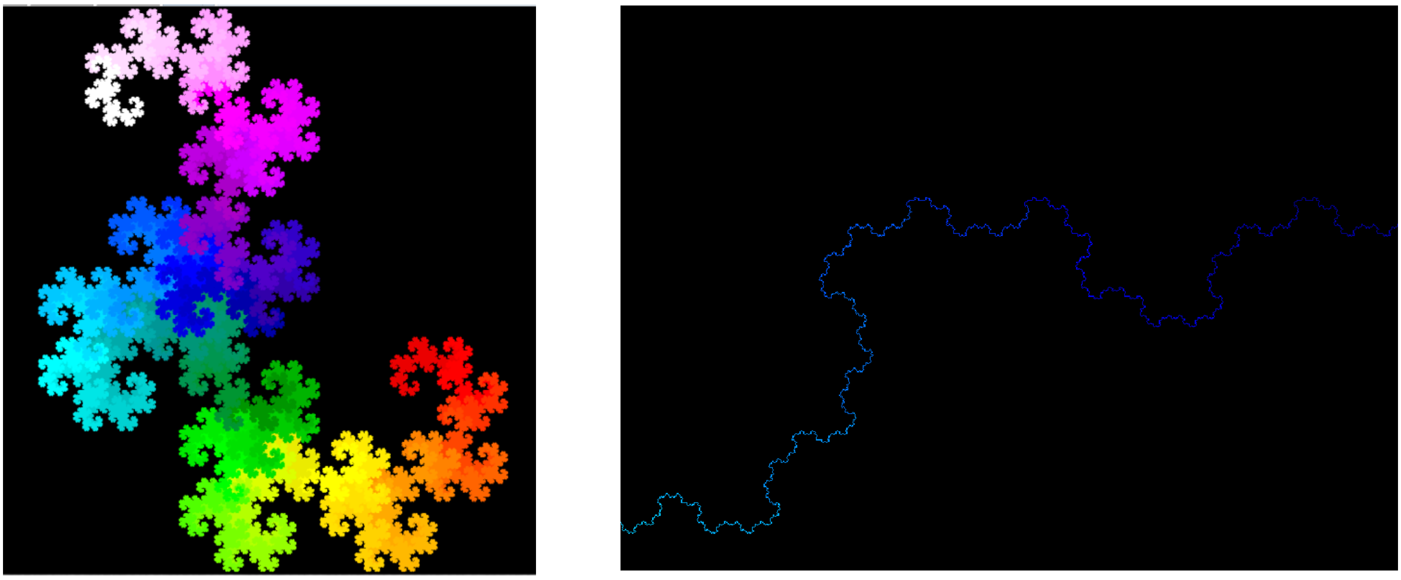


Figura 14 - Fratal Paper Folding com angulos de 90º e 130º

Permite inclusive dar zoom em determinadas partes do fractal que queiramos ver melhor, tendo como exemplo o "dragao" gerado (ângulo de 90 graus)

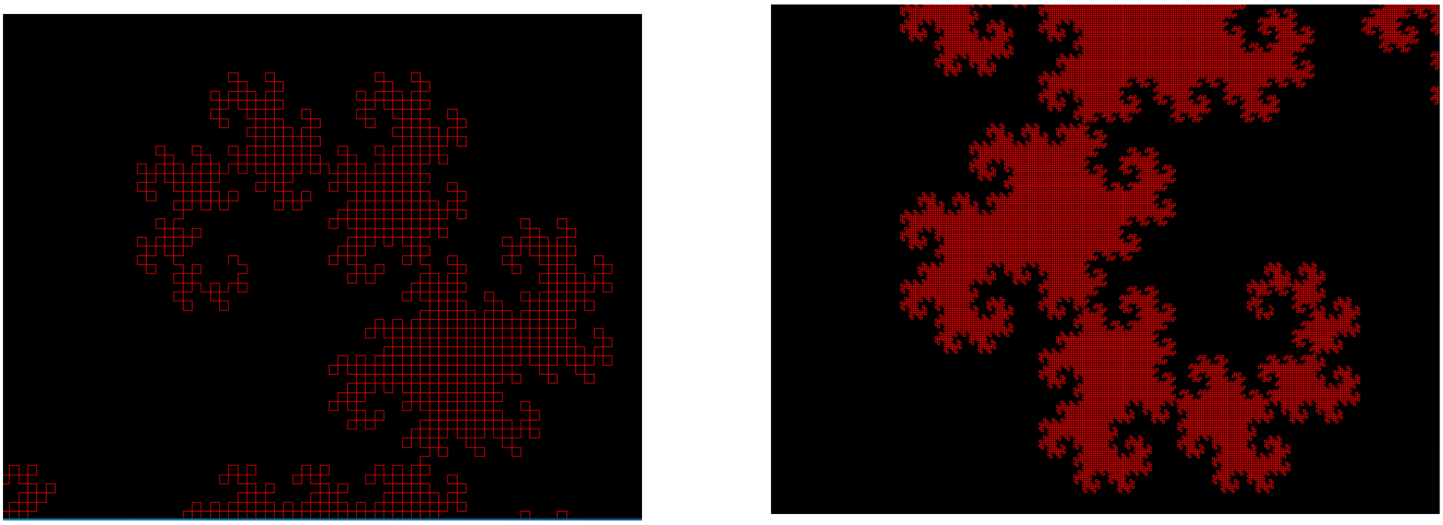


Figura 15 - zoom feito no fratal Paper Folding

A partir de uma interface simples é fácil de se utilizar, sendo possível gerar sistemas de Lindenmayer com uma gramática bem definida, e depressa ver resultados, é possível ver por exemplos fratais pré-definidos como o "fern"

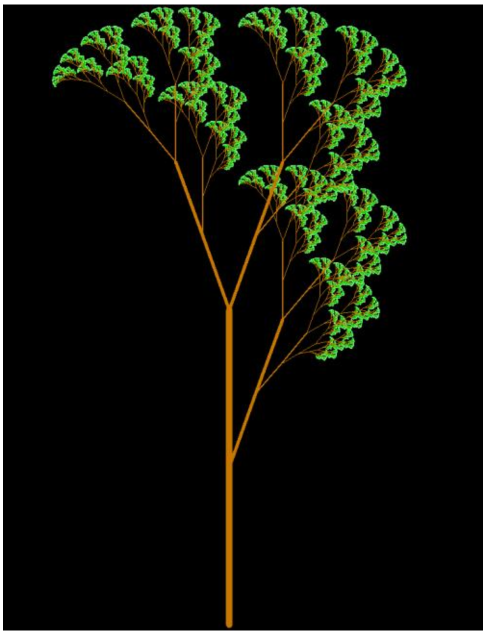
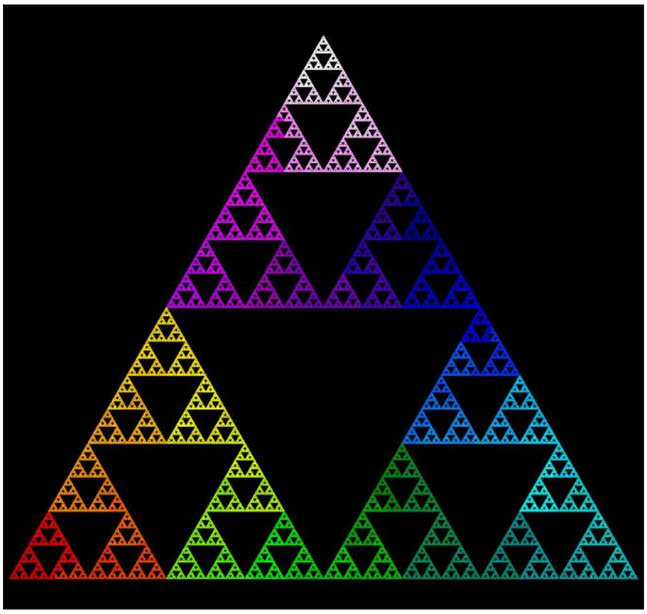


Figura 16 - diferentes imagens geradas com o fratal fern

Podemos inclusive adicionar as nossas próprias regras, ou alterar as condições iniciais, usando como exemplo, um dos fratais mais conhecidos, o triângulo de Sierpinski, em baixo é representado esse mesmo triângulo, com a sua forma original e depois, quando o axioma é igual a fh.



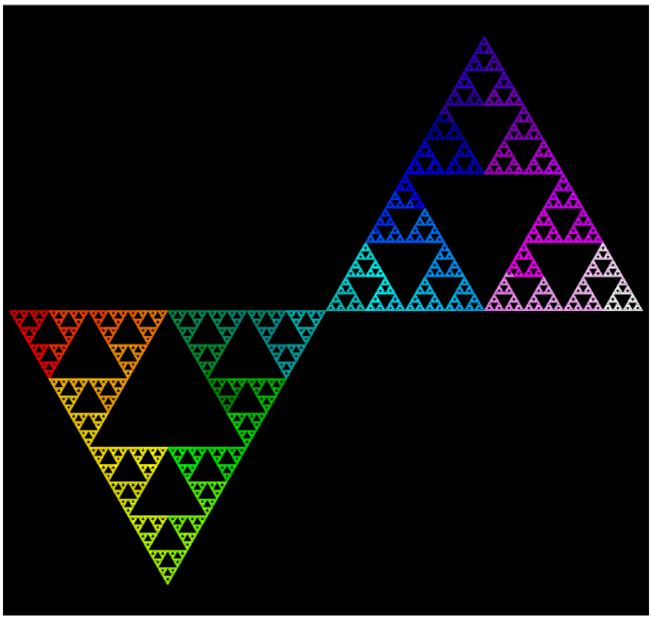


Figura 17 - à esquerda o triangulo de Sierpinski e à direita o triângulo de Sierpinski, mas com o axima igual a fh

Conclusões

Neste trabalho de casa, o grupo conseguiu aprender mais acerca de fractais e agentes autónomos. Através da execução dos diferentes exercícios proposto do trabalho de casa, o grupo conseguiu fazer quase todos os exercícios, não sendo possível a execução de todos, surgindo algumas dúvidas e complicações no meio. Apesar de algumas dificuldades sentidas, o grupo conseguiu ficar a aprender estes conceitos e consolidar esta matéria, indo com uma “bagagem” que podem vir a ser úteis para futuros trabalhos de casa e para o projeto final da disciplina.

Bibliografia

Folhas fornecidos pelo docente Arnaldo Abrantes

Sistema de LindenMayer:

- https://en.wikipedia.org/wiki/L-system
- https://web.cs.wpi.edu/gogo/courses/cs4731/assignments/ass2/
- https://cartesianfaith.com/2014/01/18/generating-artificial-plants-using-stochastic-lindenmayer-systems-with-d3-js/
- http://ai.toastbrot.ch/life/koch.php

Chaos Game:

- https://en.wikipedia.org/wiki/Chaos_game

Conjunto de Julia:

- https://en.wikipedia.org/wiki/Julia_set