Universidade Federal de Uberlândia

Faculdade de Matemática

Modelagem de L-sistemas no Âmbito Computacional do GeoGebra

Vítor Dias do Valle Tanajura
Orientando

Ana Maria Amarillo Bertone
Orientadora



Uberlândia - MG

2020

Vítor Dias do Valle Tanajura

MODELAGEM DE L-SISTEMAS NO ÂMBITO COMPUTACIONAL DO GEOGEBRA

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Uberlândia (UFU), como parte das exigências para a obtenção do título de bacharel em Matemática.

Orientadora: Profa. Dra. Ana Maria Amarillo Bertone.

Uberlândia

2020

Vítor Dias do Valle Tanajura

MODELAGEM DE L-SISTEMAS NO ÂMBITO COMPUTACIONAL DO GEOGEBRA

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Uberlândia (UFU), como parte das exigências para a obtenção do título de bacharel em Matemática.

Uberlândia, ____de_____de_____

BANCA EXAMINADORA

Profa. Dra. Ana Maria Amarillo Bertone (Orientadora)
Universidade Federal de Uberlândia (UFU)

Profa. Dra. Luciana Aparecida Alves
Universidade Federal de Uberlândia (UFU)

Profa. Dra. Lúcia Resende Pereira
Universidade Federal de Uberlândia (UFU)

Uberlândia

2020

Agradecimentos

Em primeiro lugar, a Deus, por sempre me dar forças nos momentos em que sinto que não as tenho mais, nos momentos de desânimo e nos momentos de extremo cansaço.

Aos meus amigos: Beatriz, Fernanda, Leonardo, Caroline e meus familiares: meus pais, meu irmão e minha prima, Rafaela, por sempre estarem presentes, em qualquer momento na minha vida, por aturarem minhas reclamações diárias, todavia comemorar comigo nos momentos de conquistas e realizações.

À minha orientadora, professora Ana Maria, que conduziu o trabalho com extrema paciência e dedicação, sempre disponível a compartilhar todo o seu vasto conhecimento.

Ao professor Jefferson, que criou e nos doou toda a codificação para que fosse possível a modelagem dos objetos matemáticos presentes neste estudo. Além de transmitir a mim enorme conhecimento sobre programação.

Às professoras que compõe a banca, Luciana e Lúcia, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional ao longo do curso.

Por fim, a todos que participaram, direta ou indiretamente do desenvolvimento deste trabalho de conclusão de curso, enriquecendo o meu processo de aprendizado.

Resumo

Através de modelagens matemáticas e computacionais desenvolvem-se objetos matemáticos via sistemas dinâmicos discretos conhecidos como sistemas de Lindenmayer ou L-sistemas. Entre os objetos estudados e modelados encontram-se fractais (Conjunto de Cantor e Floco de Neve de Koch), curvas de preenchimento espacial, sem auto-interseções, contínuas, e auto-similares (Curvas de Peano, Hilber e Moore, Curva do Dragão), árvores axiais de Lindenmayer, e tesselações no plano $\mathbb{R}^2$, com enfoque na conhecida tesselação de Penrose. Com o excepcional auxílio do *software* GeoGebra em conjunto com suas ferramentas poderosas de codificação, a tartaruga, utlizando a dinâmica do *Turtle Graphics* e o *JavaScript*, têm sido possíveis a visualização das dinâmicas com simulações neste ambiente computacional. Outro aspecto da visualização das modelagens é fornecido pela interação com um usuário via os *Applets* produzidos com as dinâmicas no GeoGebra. Finalizando com animações desenvolvidas através do *software* LaTeX e seu pacote `\usepackage{animate}` que, se encontram no apêndice deste manuscrito e são mostradas no arquivo de formato PDF.

Palavras-chave: GeoGebra. L-sistemas. *Turtle Graphics*.

Abstract

Through mathematical and computational modeling, objects from mathematics theories are developed via discrete dynamic systems known as Lindenmayer systems, or L-systems. Among the objects that has been studied and modeled are fractals (Cantor's Set and Koch's Snowflake), space-**F**illing, self-**A**voiding, **S**imple, and self-**S**imilar (FASS curves) (Peano, Hilber and Moore Curves, Dragon's Curve), Lindenmayer axial trees, and tessellations in the plane $\mathbb{R}^2$, focusing on the well-known Penrose tessellation. With the help of *software* GeoGebra using Turtle Graphics and JavaScript, it has been possible to create a dynamic visualization, with simulations, in this computational environment. Another aspect of the visualization is provided by the interaction a user has via the Applets that have been produced in GeoGebra. And finally in the appendix of this manuscript of the PDF file there are samples of animations developed using the `\usepackage{animate}` available in LaTeX.

Keywords: GeoGebra. L-systems. Turtle Graphics.

Lista de Figuras

Figura 1.1 A interface do GeoGebra. 6
Figura 1.2 Dois comandos da dinâmica da tartaruga. 7
Figura 1.3 Iniciando a tartaruga, de nome mat, no Geogebra. 8
Figura 1.4 Iniciando várias tartarugas com diferentes nomes. 8
Figura 1.5 Tartaruga, de nome mat, andando 4 unidades para frente. 9
Figura 1.6 Tartaruga, de nome mat, andando 4 unidades para trás. 10
Figura 1.7 Tartaruga, de nome mat, girando seu corpo em um ângulo de $\frac{\pi}{2}$ radianos no sentido horário. 11
Figura 1.8 Tartaruga, de nome mat, girando seu corpo em um ângulo de $\frac{\pi}{2}$ radianos no sentido anti-horário. 11
Figura 1.9 Processo de criação de um quadrado utilizando os comandos da tartaruga. 13
Figura 1.10 Tartaruga, de nome mat, transladada da origem do plano cartesiano para as coordenadas $(x, y) = (3, 4)$. 14
Figura 1.11 Nó n_0 rodeado de polígonos regulares sem haver espaços e sobreposições. 17
Figura 1.12 Triângulo Equilátero de lado $l = 1$. 18
Figura 1.13 Primeira fileira horizontal composta por 15 triângulos equiláteros de lado $l = 1$ voltados para cima. 18
Figura 1.14 Primeira fileira horizontal composta por 15 triângulos equiláteros de lado $l = 1$ voltados para cima, em conjunto com, a fileira consecutiva de triângulos voltados para baixo. 19
Figura 1.15 Ponto C de retorno da tartaruga para realização da próxima fileira de triângulos equiláteros voltados para cima e para baixo. 19
Figura 1.16 Triângulo Equilátero de lado l e altura $h = \frac{l\sqrt{3}}{2}$. 19
Figura 1.17 Tesselação Unicelular Regular formada por Triângulos Equiláteros. 20
Figura 1.18 Quadrado de lado $l = 1$. 21
Figura 1.19 Primeira fileira horizontal composta por 15 quadrados lado $l = 1$. 21
Figura 1.20 Tesselação Unicelular Regular formada por Quadrados. 22
Figura 1.21 Hexágono Regular de lado $l = 1$. 23
Figura 1.22 Primeira fileira horizontal composta por 15 hexágonos regulares de lado $l = 1$. . . . 24

Figura 1.23 Pontos A, B e C de retorno da tartaruga após a realização das primeira, segunda e terceira fileiras horizontais respectivamente. 24
Figura 1.24 Hexágono Regular de lado l. 24
Figura 1.25 Tesselação Unicelular Regular formada por Hexágonos Regulares. 26
Figura 1.26 Botões programáveis criados para auxiliar a execução dos comandos *JavaScript* pela tartaruga. 27
Figura 1.27 Janela de Propriedades de um botão criado previamente. 28

Figura 2.1 Interface do GeoGebra com a modelagem do L-sistema dado, com ângulo $\delta = \frac{\pi}{2}$, das 1^a, 2^a e 3^a gerações, com suas respectivas substituições. 35

Figura 3.1 Modelagem do conjunto de Cantor da 1^a geração até a 6^a geração, utilizando a tartaruga do GeoGebra com o *JavaScript*. 37
Figura 3.2 Interface do GeoGebra com a modelagem da primeira geração ($n = 1$ na "Janela e Álgebra") do floco de neve de Koch. 44
Figura 3.3 Interface do GeoGebra com a modelagem da quarta geração ($n = 4$ na "Janela e Álgebra") do floco de neve de Koch. 44
Figura 3.4 Interface do GeoGebra com a modelagem da sexta geração ($n = 6$ na "Janela e Álgebra") do floco de neve de Koch. 45

Figura 4.1 Modelagem Matemática e Computacional da curva de Peano com a interface do GeoGebra na 4^a geração ($n = 4$ na "Janela de Álgebra"). 50
Figura 4.2 Modelagem Matemática e Computacional da curva de Hilbert com a interface do GeoGebra na 6^a geração ($n = 6$ na "Janela de Álgebra"). 52
Figura 4.3 Modelagem Matemática e Computacional da curva de Moore com a interface do GeoGebra na 5^a geração ($n = 5$ na "Janela de Álgebra"). 55
Figura 4.4 Simulação da primeira à terceira iteração na formação da curva do Dragão. 58
Figura 4.5 Ambiente do GeoGebra com todos os elementos de programação e iteração com usuário. Note que $n = 11$ na "Janela de Álgebra" representa a 11^a geração da curva do Dragão e com passo de tamanho escolhido $length = 1$. 59

Figura 5.1 O esquema de uma L-árvore enraizada. 61
Figura 5.2 O colchete [armazena e empilha uma posição e angulação da tartaruga. Estes dados só serão desempilhados quando chegar no colchete]. O processo é análogo aos colchetes [ ] e [ ] 66
Figura 5.3 Interface do GeoGebra com a modelagem da quinta geração ($n = 5$ na "Janela de Álgebra") de uma árvore de Lindenmayer. 66

Figura 5.4 Interface do GeoGebra com a modelagem da sexta geração ($n = 6$ na "Janela de Álgebra") de outra árvore de Lindenmayer. 69
Figura 5.5 Interface do GeoGebra com a modelagem da quinta geração ($n = 5$ na "Janela de Álgebra") de outra árvore de Lindenmayer. 72

Figura 6.1 Interface do GeoGebra com a modelagem de uma tesselação da oitava geração ($n = 8$ na "Janela de Álgebra"). 77
Figura 6.2 Roger Penrose no saguão do *Mitchell Institute for Fundamental Physics and Astronomy, Texas A&M University*, em pé no chão com a tesselação de Penrose modelada nesta seção. 78
Figura 6.3 Interface do GeoGebra com a modelagem da tesselação de Penrose da primeira geração ($n = 1$ na "Janela de Álgebra"). 82
Figura 6.4 Interface do GeoGebra com a modelagem da tesselação de Penrose da segunda geração ($n = 2$ na "Janela de Álgebra"). 82
Figura 6.5 Interface do GeoGebra com a modelagem da tesselação de Penrose da terceira geração ($n = 3$ na "Janela de Álgebra"). 83
Figura 6.6 Interface do GeoGebra com a modelagem da tesselação de Penrose da quarta geração ($n = 4$ na "Janela de Álgebra"). 83
Figura 6.7 Modelagem da tesselação de Penrose da primeira geração ou axioma ($n = 1$ na "Janela de Álgebra"). 87
Figura 6.8 Modelagem da tesselação de Penrose da segunda geração ($n = 2$ na "Janela de Álgebra"). 87
Figura 6.9 Modelagem da tesselação de Penrose da terceira geração ($n = 3$ na "Janela de Álgebra"). 88
Figura 6.10 Modelagem da tesselação de Penrose da quinta geração ($n = 5$ na "Janela de Álgebra"). 88

Lista de Tabelas

Tabela 1.1 Métodos especiais do *JavaScript*, os quais pertencem ao conjunto de comandos *ggbApplet* utilizados no GeoGebra para trabalhar com a tartaruga. 31

Tabela 1.2 Outros comandos do GeoGebra que através do *JavaScript* são utilizados na modelagem. 32

Tabela 4.1 As ternas G_P, G_H e G_M correspondentes à curva de Peano, Hilbert e Moore, respectivamente. 47

Sumário

Agradecimentos **i**

Resumo **ii**

Abstract **iii**

Lista de Figuras **iv**

Lista de Tabelas **vii**

Introdução **1**

1 As Ferramentas da Modelagem Matemática: GeoGebra, Tartaruga e *JavaScript* **5**

1.1 O GeoGebra 5

1.2 A Tartaruga do GeoGebra 6

1.2.1 Pequeno Manual para Utilização da Tartaruga do GeoGebra 7

1.2.2 Tesselação Formada por Triângulos Equiláteros 17

1.2.3 Tesselação Formada por Quadrados 20

1.2.4 Tesselação Formada por Hexágonos Regulares 22

1.3 *JavaScript* 26

2 Sistemas de Lindenmayer **33**

2.1 Caracterização dos L-sistemas 33

2.2 Ilustração de um L-sistema 34

3 Alguns Fractais Modelados via L-sistemas **36**

3.1 O Conjunto de Cantor 37

3.2 O Floco de Neve de Koch 40

4 L-sistemas e as Curvas FASS **46**

4.1 As Curvas FASS 46

4.2 Peano, Hilbert e Moore: Três Curvas FASS Entrelaçam Três Mentes Brilhantes 47

4.3 A Curva do Dragão 55

5 Árvores de Lindenmayer **60**

5.1 A Modelagem Botânica de Lindenmayer 60

5.2 Simulações de Árvores de Lindenmayer 61

6 Tesselações via L-sistemas **73**

6.1 Tesselações: Alicerce Teórico 73

6.2 Uma Primeira Tesselação Utilizando um L-sistema 74

6.3 Esbarrando com o Primeiro Problema Técnico: A Tesselação de Penrose 78

Conclusão **89**

Apêndice **90**

Referências Bibliográficas **93**

Índice Remissivo **97**

Introdução

A instigação provocada pelas diversas aplicações da matemática no mundo real vistas pelo orientando na disciplina de Modelagem Matemática, causou necessidade e curiosidade de implementar outros objetos matemáticos, com o auxílio da tecnologia representada pelo GeoGebra, um *software* livre e suas ferramentas de codificação. Um dos trabalhos propostos, na disciplina supracitada, tinha como objetivo implementar a famosa dança do requebrado das abelhas que foi desenvolvida, totalmente, no *software* GeoGebra. O que induz a imaginar um assunto similar para a produção de modelagens no âmbito computacional do GeoGebra como tópico central deste trabalho de conclusão.

Desta maneira, foi proposto de início, o estudo e aplicação sobre a padronização do formato de teias de aracnídeos. Entretanto, viu-se em uma encruzilhada pois, ao mesmo tempo que a ideia de tentar buscar explicações e métodos para criação destes formatos fosse algo estimulante, não havia material o suficiente para o mesmo. Diante deste cenário, foi proposto, em seguida, a modelagem matemática e computacional de objetos matemáticos que podem ser descritos por sistemas de Lindenmayer, como fractais, curvas de preenchimento, tesselações (mais especificamente, a tesselação de Penrose) e, por fim, as próprias L-árvores resultados destes sistemas dinâmicos discretos. Com a quantidade exorbitante de material disponível e o suporte gráfico e computacional do GeoGebra em conjunto com as suas ferramentas, fez viável esta escolha de modelagem matemática e computacional.

A modelagem matemática obtida através de um sistema de Lindenmayer, conhecido como L-sistema, foi originalmente proposta pelo biólogo Aristid Lindenmayer em 1968 [44] como base para uma teoria axiomática do desenvolvimento biológico. No ano de 1990, Aristid Lindenmayer consolidou uma parceria científica com Przemysław Prusinkiewicz, um famoso cientista da computação, cooperação que resultou na publicação do seu fascinante livro intitulado *The Algorithmic Beauty of Plants*. A proposta do livro é realizar modelos botânicos e inicia-se com a simulação de filamentos da bactéria *Anabaena catenula* e de várias algas [44].

O objetivo básico dos L-sistemas é definir objetos complexos substituindo sucessivamente partes de um objeto simples e usando um conjunto de regras de produção recursiva, técnica também chamada de reescrita ou substituição [44]. Além disso, segundo Lindenmayer e Prusinkiewicz:

> [...] L-sistemas foram utilizados, primeiramente para determinar a topologia ramificada de plantas modeladas [...] L-sistemas poderiam gerar as intrigantes e complexas curvas conhecidas, hoje, como fractais [...] L-sistemas geram as clássicas curvas de preenchimento espacial [44].

Os L-sistemas encontraram diversas aplicações desde sua criação. Para mencionar alguns trabalhos a partir da década dos anos 90, encontram-se aplicações diversas: a de Boer (1990) [14] na análise do processo de desenvolvimento de camadas de células; em 1993, Hamilton [19] que empregou os L-sistemas na geração de neurônios artificiais, seguido nessa linha de pesquisa por McCormick e Mulchandani (1994) [34], Kalay, Parnas e Shamir (1995) [25]; logo após, em 1995, Cowell [10] promove o reconhecimento de padrões na identificação de placas de automóveis, utilizando L-sistemas; no mesmo ano, Holliday e Samall [23] propuseram o reconhecimento de formas vegetais como folhas, plantas menores e árvores em visão computacional; mais recentemente, no ano de 2001, Hornby e Pollack [24] utilizaram os L-sistemas para gerar criaturas virtuais para ambientes de simulação; após alguns anos, em 2006, Manousakis [33] aplicou os L-sistemas na geração de estruturas de arranjos musicais. A esquemática proposta por Lindenmayer e Prusinkiewicz em seu livro foi a fonte de inspiração das modelagens de objetos matemáticos diversos.

Uma sucinta menção à teoria dos L-sistemas, inclui a definição de uma estrutura matemática composta por:

1. Um alfabeto que é um conjunto de símbolos matemáticos. Os símbolos utilizados neste estudo são: letras do alfabeto comum que representam as variáveis do alfabeto e, em geral, são denotadas em maiúscula; os símbolos matemáticos +, -, [,]; medidas de comprimento ou de amplitude. Dependendo da modelagem matemática computacional a ser desenvolvida, as compontentes do alfabeto são substituíveis ou fixas;

2. Um axioma que é um conjunto ordenado não vazio composto por símbolos do alfabeto;

3. Um conjunto de regras de produção que determinam a dinâmica do L-sistema. Matematicamente, estas regras são representadas por um par ordenado cuja primeira componente é um elemento do alfabeto e a segunda componente é um conjunto ordenado de símbolos do alfabeto, sendo que esta pode ser um conjunto vazio.

Os L-sistemas são modelados matematicamente e computacionalmente de várias maneiras. A forma escolhida nesta pesquisa é a da dinâmica conhecida como *turtle graphics* (veja Seção 1.2), metodologia que introduz um autômato no plano cartesiano representado pela imagem gráfica de uma tartaruga. À posição da tartaruga no plano, representada pelas suas coordenadas, acrescenta-se a direção de seu corpo, determinado pelo ângulo α que forma com o eixo positivo Ox no sentido anti-horário, caracterizando a posição da tartaruga por uma terna (x, y, α). A tartaruga realiza movimentos para frente e para trás, além de giros nos sentidos horário e anti-horário tomando seu próprio corpo como eixo [36].

O meio computacional escolhido para o desenvolvimento da modelagem dos L-sistemas é o *software* GeoGebra [22], o qual foi introduzido pelo professor Markus Hohenwarter da Universidade de Salzburgo (Áustria) em 2001. A escolha vem do fato do GeoGebra ser, além de um *software* matemático, também um *software* livre de simples codificação com uma interface fácil de se operar. Além da tartaruga, outra

importante ferramenta que esse *software* oferece, de forma a simplificar a modelagem, é o *JavaScript*, que, neste estudo, auxilia na manipulação dos movimentos da tartaruga. De fato, seus movimentos são realizados através de loops necessários, também, para a modelagem dos L-sistemas.

O *JavaScript*, segundo o site MDN, por *Mozilla Contributors* (2020) [49], é uma linguagem de programação que permite criar conteúdo que se atualiza dinamicamente, controlar multimídias, imagens animadas, e tudo o mais que há de interessante. A conexão é dada atráves da janela de programação avançada, na qual é escrita a codificação que permite a comunicação com a "Janela de Visualização" do GeoGebra. Esse "diálogo" é realizado através do grupo de comandos *ggbApplet* que se comunica com os objetos criados no GeoGebra por meio da linguagem *GGBScript*. Assim, este fluxo objeto-*ggbApplet*-*JavaScript* permite a interação entre o código da linguagem *JavaScript* e os elementos iterativos do *software* GeoGebra.

Em vista do exposto, o objetivo geral deste trabalho é a modelagem matemática através dos L-sistemas utilizando como meio computacional o *software* GeoGebra. A organização do trabalho segue a evolução gradativa das etapas desenvolvidas durante o estudo: desde o código mais simples (conjunto de Cantor) até o mais complexo (tesselação de Penrose). Os tópicos abordados envolvem os seguintes assuntos:

1. L-sistemas associados a dois fractais, o conjunto de Cantor e a curva que determina o floco de neve de Koch, desenvolvidos no Capítulo 3;

2. Curvas de preenchimento espacial FASS, em particular, a de Peano, Hilbert, Moore e a conhecida como curva do Dragão. Este tópico é exposto no Capítulo 4;

3. Modelagem de árvores axiais propostas por Lindenmayer, detalhadas no Capítulo 5;

4. L-sistemas associados a tesselações do plano $\mathbb{R}^2$, entre elas, a mais famosa devido a Roger Penrose. As curiosidades matemáticas dessa tesselação assim como as dificuldades encontradas na sua implementação são explicitadas no Capítulo 6.

Todos esses assuntos expostos com suas respectivas referências bibliográficas.

Por outro lado, expõe-se no Capítulo 1 justificativas do uso do GeoGebra, da utilização do *turtle graphics*, representado com a tartaruga do GeoGebra, assim como um manual escrito neste estudo para facilitar a iniciação do usuário nessa modalidade. Completa-se o Capítulo 1 com modelagens de tesselações unicelulares regulares que ilustram a técnica da tartaruga no ambiente do *software* e como contribuição para uma audiência interessada no assunto.

No capítulo de Conclusão deste trabalho apresenta-se uma análise dos resultados durante o processo de construção dos assuntos abordados. Principalmente na evolução dos aspectos críticos, nos avanços nas técnicas de escrita e de vários aspectos relacionados com a academia e a profissão. Além da confirmação dos objetivos e estabelecimentos de trabalhos futuros.

Salienta-se que todas as figuras contidas neste trabalho foram construídas utilizando o *software GeoGebra*, exceto por uma imagem do prêmio nobel de física 2020, Roger Penrose, extraído de uma fonte citada.

Assim como todas as citações recuadas e as frases iniciais (*quotes*) de cada capítulo são traduções do inglês, exceto a do Seymour Papert no Capítulo 1.

Encerra-se este trabalho com um apêndice com simulações dos L-sistemas modelados feitas para o arquivo PDF utilizando o pacote do *software* LaTeX denominado *animate*. Esta modelagem computacional ilustra dinamicamente, em interação com o leitor, os resultados obtidos ao longo deste estudo.

Capítulo 1

As Ferramentas da Modelagem Matemática: GeoGebra, Tartaruga e *JavaScript*

"... A ideia básica do GeoGebra é reunir geometria, álgebra e cálculo,
que outros softwares tratam separadamente,
em um único pacote fácil de usar para aprender e ensinar matemática
desde o nível elementar até a universidade."
Markus Hohenwarter [21] (2007).

1.1 O GeoGebra

O instrumento computacional para o desenvolvimento deste trabalho é o *software* GeoGebra [22], o qual foi introduzido pelo professor Markus Hohenwarter da Universidade de Salzburgo (Áustria) em 2001.

O GeoGebra foi desenvolvido para ensino-aprendizagem e é classificado como um *software* livre de matemática, por ser baseado na GNU *General Public License* (Licença Pública Geral). A GNU GPL é a licença com maior utilização por parte de projetos de *softwares* livres. Baseia-se em quatro liberdades: liberdade de executar o programa, liberdade de estudar como o programa funciona e adaptá-lo para as necessidades, liberdade de redistribuir cópias e liberdade de aperfeiçoar o programa.

Além do GeoGebra ter a vantagem de ser um *software* livre, possui a qualidade de simples codificação, com uma interface fácil de se operar. Ainda assim, com muitos recursos e ferramentas poderosos para a criação de materiais didáticos, como *Applets* para iteração com o usuário e utilização em páginas *web* através da conversão para a linguagem *html*. Outra vantagem é o imediato acesso e visualização aos resultados no processo de construção.

A versão do *software* utilizada é: GeoGebra Classic 5.0, que permite a programação com a linguagem *JavaScript* interagindo com a linguagem do próprio GeoGebra, *ggbApplet*. Sua interface pode ser vista na Figura 1.1. À esquerda, nesta figura, se encontra a "Janela de Álgebra", no centro, a "Janela de Visualização", à direita, a janela de programação avançada e na parte inferior se encontra o "Campo de Entrada" no

qual é introduzida parte da codificação.

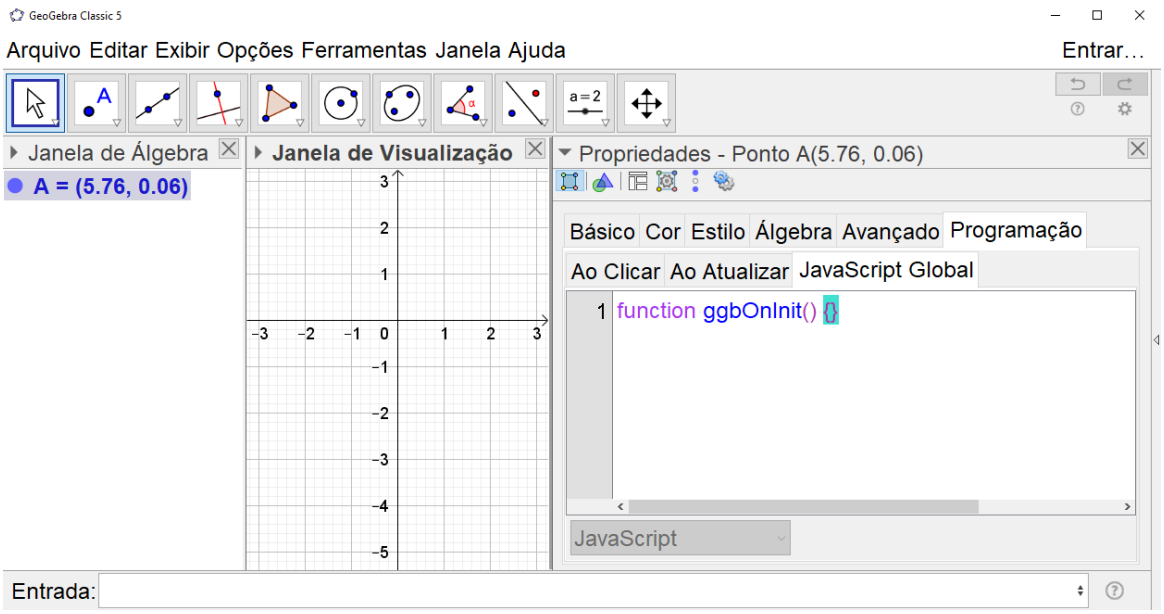


Figura 1.1: A interface do GeoGebra.

1.2 A Tartaruga do GeoGebra

Uma das ferramentas do *software* GeoGebra pouco conhecida, é a tartaruga que obedece comandos de forma similar à linguagem de programação interpretativa LOGO [36], introduzida por Seymour Papert. De fato, Papert foi um matemático e educador estadunidense do *Massachusetts Institute of Technology* (MIT). Escreveu o famoso livro *Mindstorms: Children, Computers and Powerful Ideas*, em Nova Iorque, em 1980, que foi traduzido e relançado pela editora Brasiliense, em 1985, com o título: Logo: Computadores e Educação [36]. A partir disso, Papert propôs, na década de 60, juntamente com Wally Feurzeig e Cynthia Solomon, ambos cientistas da computação, a linguagem de programação LOGO.

Em seu livro, Papert dá uma breve definição sobre a tartaruga que ele utiliza em seus estudos:

> [...] é um animal cibernético controlado pelo computador. Ela existe dentro das miniculturas cognitivas do "ambiente LOGO", sendo LOGO a linguagem computacional que usamos para nos comunicar com a tartaruga. Essa Tartaruga serve ao único propósito de ser fácil de programar e boa para se pensar [36].

No GeoGebra, pode-se definir a tartaruga como uma imagem gráfica associada à interação entre usuário e todas as ferramentas desse *software* livre. Sua movimentação é baseada na dinâmica computacional do *turtle graphics*, proposta por Papert [36]. Segundo Papert [36] (com respeito à tartaruga) "[...] sua principal função é servir como modelo para outros objetos, ainda a serem inventados". Observe-se que, na realidade, a tartaruga do GeoGebra não é uma ferramenta exclusiva do GeoGebra .

Precisando mais detalhes do método computacional do *turtle graphics*, esta metodologia consiste em um autômato bidimensional cujo estado atual é definido por coordenadas cartesianas e uma direção. O ângulo formado pela cabeça da tartaruga no sentido anti-horário com respeito ao eixo Ox é que determina essa direção. Descrevendo ainda mais a metodologia se o estado atual da tartaruga é determinado pela posição $(x_0,\ y_0)$ no plano cartesiano e, denotando por α a direção supracitada, o estado atual da tartaruga é $(x_0,\ y_0,\ \alpha)$. Se a tartaruga efetuar um passo para frente de comprimento d, então a posição da tartaruga se altera de $(x_0,\ y_0,\ \alpha)$, para $x_1 = x_0 + d\cos(\alpha)$ e $y_1 = y_0 + d\,\mathrm{sen}(\alpha)$, ou de $(x_0,\ y_0,\ \alpha)$ para $(x_0,\ y_0,\ \alpha+\delta)$, respectivamente. Isto é mostrado na Figura 1.2 (a). Se a tartaruga girar um ângulo δ no sentido horário, o próximo estado da tartaruga é $(x_0,\ y_0,\ \alpha-\delta)$. Da mesma forma, se a tartaruga girar um ângulo δ no sentido anti-horário, o seu próximo estado da tartaruga é $(x_0,\ y_0,\ \alpha+\delta)$. Isto é mostrado na Figura 1.2 (b).

(a) Movimento para frente. (b) Giro anti-horário.

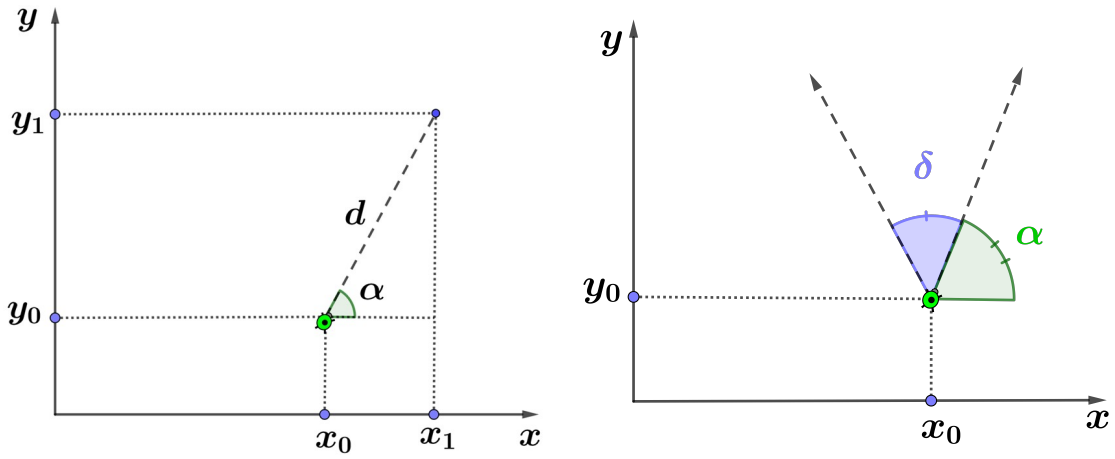


Figura 1.2: Dois comandos da dinâmica da tartaruga.

Desta forma, a função básica da tartaruga do GeoGebra é traçar segmentos ou fazer giros em torno da sua posição, à medida que se desloca no plano cartesiano.

Seu estado inicial é na origem de coordenadas $(0, 0)$ com seu corpo formando um ângulo nulo com o eixo positivo Ox. Isto a torna extremamente útil na construção de desenhos livres e de uma grande variedade de figuras geométricas.

Na próxima seção é apresentado um pequeno manual de comandos básicos da tartaruga para que um usuário interessado consiga compreender, de uma maneira simples, rápida e eficiente, como começar a interagir com essa ferramenta do GeoGebra.

1.2.1 Pequeno Manual para Utilização da Tartaruga do GeoGebra

Esta seção tem como objetivo guiar o leitor para o uso básico da tartaruga no GeoGebra. Seguindo as instruções, o usuário poderá desenvolver figuras geométricas desde as mais básicas até as mais complexas dependendo de sua criatividade. Neste momento, supõe-se que o usuário está habituado ao uso básico do GeoGebra.

O manual será dividido em dois tópicos a fim de mantê-lo o mais organizado possível: **Iniciando a Tartaruga no GeoGebra** e **Movimentando a Tartaruga no GeoGebra**.

1. **Iniciando a Tartaruga no GeoGebra**

   Para iniciá-la no GeoGebra, digite no "Campo de Entrada" o comando $Tartaruga(\ )$. Pode-se renomeá-la com o nome que o usuário desejar digitando antes do comando $Tartaruga(\ )$, por exemplo, $mat = Tartaruga(\ )$. Assim, o nome da nova tartaruga será mat. Diferenciando com um nome pode-se criar tantas tartarugas quanto se queira. As Figuras 1.3 e 1.4 mostram esse procedimento.

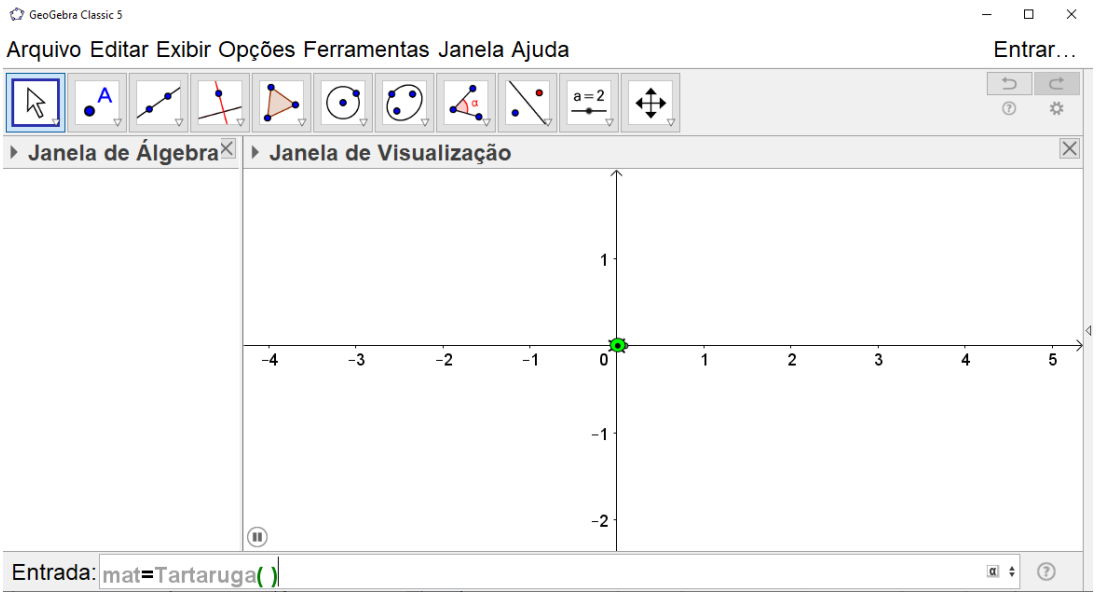


Figura 1.3: Iniciando a tartaruga, de nome mat, no Geogebra.

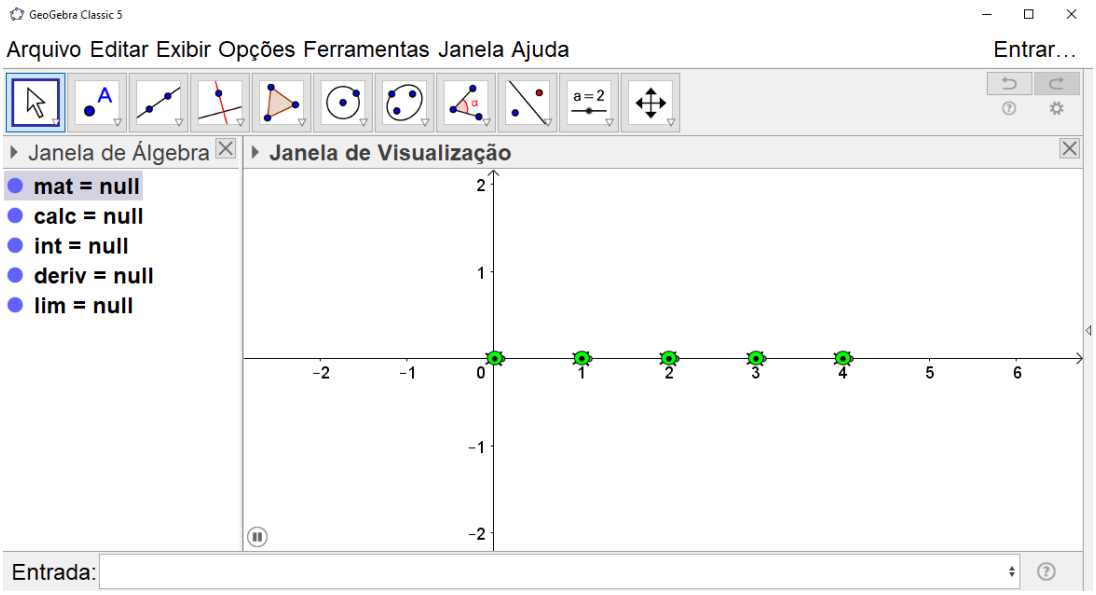


Figura 1.4: Iniciando várias tartarugas com diferentes nomes.

2. **Movimentando a Tartaruga no GeoGebra**

Após iniciar a tartaruga, percebe-se que aparece um botão de "*play*"  no canto inferior esquerdo da "Janela de Visualização" logo acima do "Campo de Entrada". Este botão possui a finalidade de iniciar os movimentos da tartaruga após o usuário digitar no "Campo de Entrada" seus comandos. Os possíveis movimentos que ela pode executar serão descritos a seguir.

- $TartarugaIrParaFrente(< Tartaruga >, < Distância >)$:

  Este comando faz a tartaruga andar para frente o número de unidades que o usuário desejar, ao mesmo tempo que desenha segmentos de reta.

  Basta colocar o nome da tartaruga que o usuário definiu no início do código em $< Tartaruga >$ e o número de unidades que a tartaruga deve andar para frente em $< Distância >$.

  Por exemplo, ao digitar $TartarugaIrParaFrente(mat, 4)$, a tartaruga chamada mat andará 4 unidades para frente, como mostra na Figura 1.5.

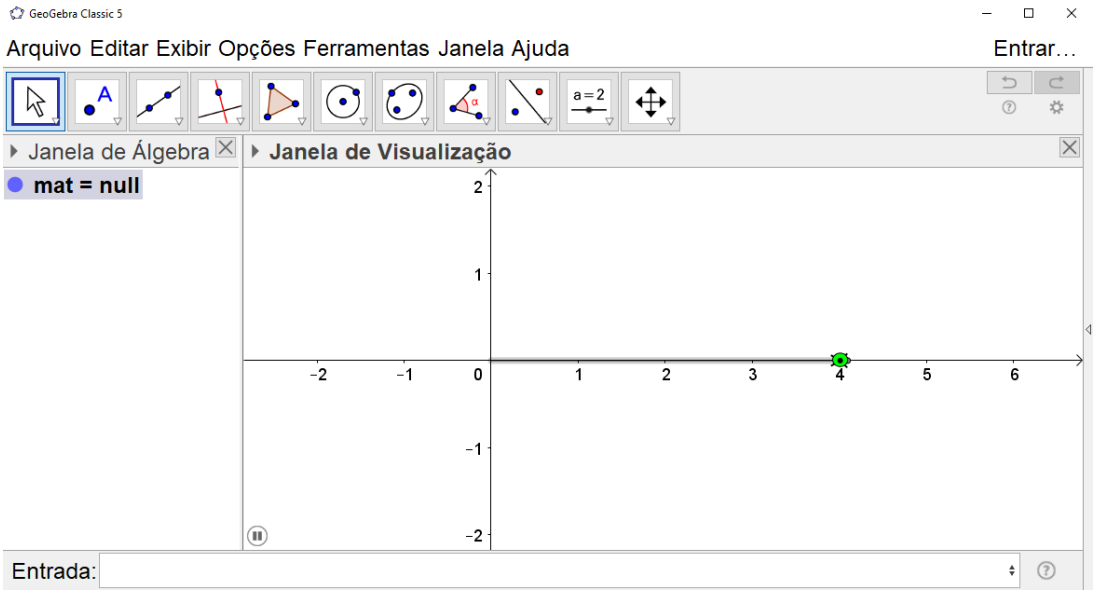


Figura 1.5: Tartaruga, de nome mat, andando 4 unidades para frente.

- $TartarugaIrParaTrás(< Tartaruga >, < Distância >)$:

  Este comando faz a tartaruga andar para trás o número de unidades que o usuário desejar, ao mesmo tempo que desenha segmentos de reta.

  Basta colocar o nome da tartaruga que o usuário definiu no início do código em $< Tartaruga >$ e o número de unidades que a tartaruga deve andar para trás em $< Distância >$.

  Por exemplo, ao digitar $TartarugaIrParaTrás(mat, 4)$, a tartaruga chamada mat andará 4 unidades para trás, como mostra na Figura 1.6.

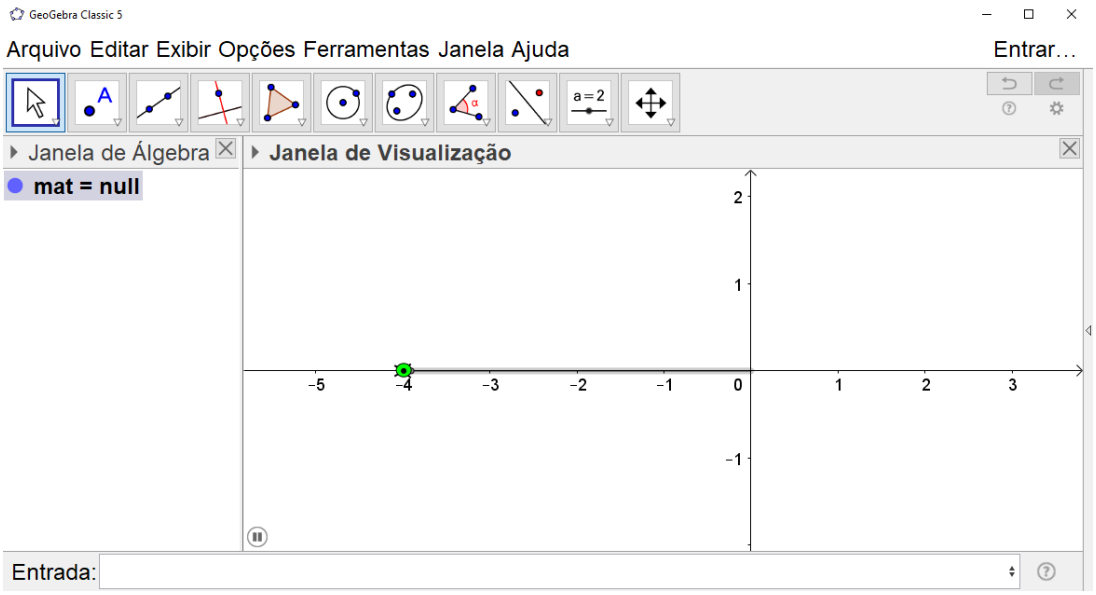


Figura 1.6: Tartaruga, de nome mat, andando 4 unidades para trás.

- $TartarugaNãoDesenhar(< Tartaruga >)$:

  Este comando permite que a tartaruga não trace segmentos de reta ao se movimentar quando os comandos $TartarugaIrParaFrente(< Tartaruga >, < Distância >)$ e/ou $TartarugaIrParaTrás(< Tartaruga >, < Distância >)$ forem utilizados.

  Basta colocar o nome da tartaruga que o usuário definiu no início do código em $< Tartaruga >$.

- $TartarugaDesenhar(< Tartaruga >)$:

  Este comando permite que a tartaruga trace segmentos de reta ao se movimentar quando os comandos $TartarugaIrParaFrente(< Tartaruga >, < Distância >)$ e/ou $TartarugaIrParaTrás(< Tartaruga >, < Distância >)$ forem utilizados.

  Basta colocar o nome da tartaruga que o usuário definiu no início do código em $< Tartaruga >$.

  Note que só há necessidade de utilizar o comando $TartarugaDesenhar(< Tartaruga >)$, após o uso do comando $TartarugaNãoDesenhar(< Tartaruga >)$.

- $TartarugaIrParaDireita(< Tartaruga >, < \hat{A}ngulo >)$:

  Este comando faz com que a tartaruga gire um ângulo qualquer no sentido horário.

  Basta colocar o nome da tartaruga que o usuário definiu no início do código em $< Tartaruga >$ e o ângulo (em radianos ou graus) em $< \hat{A}ngulo >$.

  Por exemplo, ao digitar $TartarugaIrParaDireita(mat, pi/2)$, a tartaruga chamada mat girará seu corpo em um ângulo de $\dfrac{\pi}{2}$ radianos no sentido horário, como mostra na Figura 1.7.

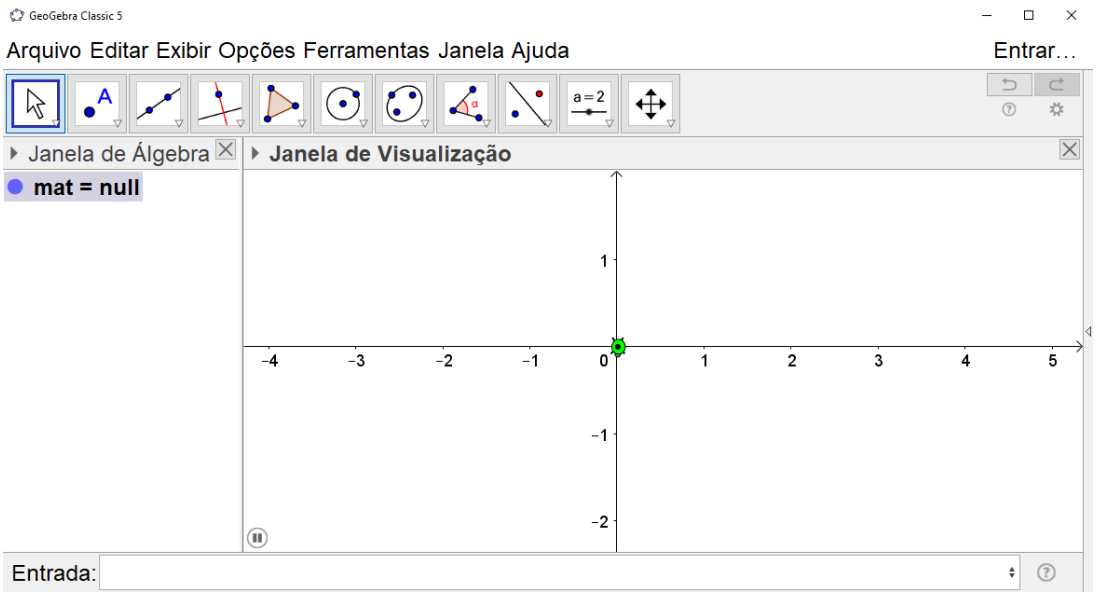


Figura 1.7: Tartaruga, de nome mat, girando seu corpo em um ângulo de $\frac{\pi}{2}$ radianos no sentido horário.

- $TartarugaIrParaEsquerda(< Tartaruga >, < \hat{A}ngulo >)$:

  Este comando faz com que a tartaruga gire um ângulo qualquer no sentido anti-horário.

  Basta colocar o nome da tartaruga que o usuário definiu no início do código em $< Tartaruga >$ e o ângulo (em radianos ou graus) em $< \hat{A}ngulo >$.

  Por exemplo, ao digitar $TartarugaIrParaEsquerda(mat, pi/2)$, a tartaruga chamada mat girará seu corpo em um ângulo de $\frac{\pi}{2}$ radianos no sentido anti-horário, como mostra na Figura 1.8.

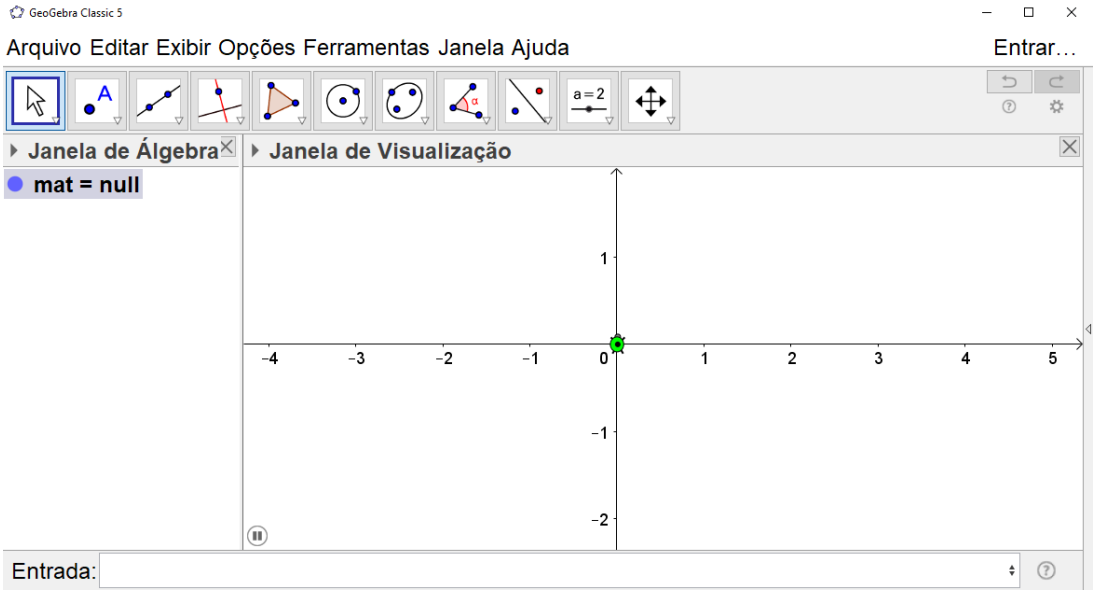


Figura 1.8: Tartaruga, de nome mat, girando seu corpo em um ângulo de $\frac{\pi}{2}$ radianos no sentido anti-horário.

Note que, nos comandos $TartarugaIrParaDireita(< Tartaruga >, < \hat{A}ngulo >)$ e $TartarugaIrParaEsquerda(< Tartaruga >, < \hat{A}ngulo >)$ a tartaruga não se movimenta,

apenas gira seu corpo.

Os próximos três comandos não são exclusivos ao uso da tartaruga no GeoGebra.

- $Repetir(< Número >, < Programação >, < Programação >, ...)$:

  Este comando possui a função de reproduzir uma grande quantidade de programações, que se repetem, um número determinado de vezes.

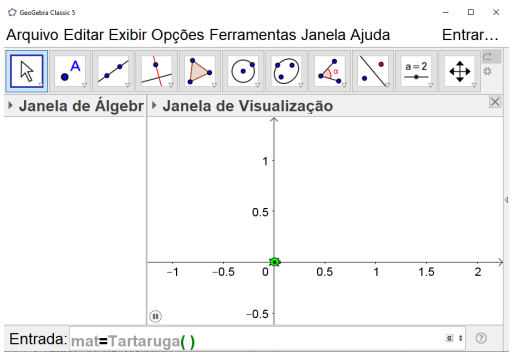
  Para utilizar o comando $Repetir$, basta colocar um número natural em $< Número >$ que representa a quantidade de vezes que devem ser executadas as programações (inseridas em $< Programação >$), na mesma ordem em que se encontram.

  No uso da tartaruga, este comando serve para otimizar tanto sua movimentação quanto o trabalho e tempo que o usuário teria em digitar comando por comando para realizar determinado procedimento.

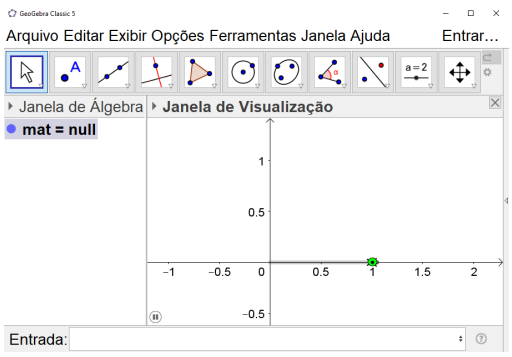
  Por exemplo, suponha que o usuário queira desenhar um quadrado de lado igual a uma unidade com a tartaruga no plano xy com $x \geq 0$ e $y \geq 0$. Para fazer isso, basta seguir o passo a passo abaixo, sempre lembrando de digitar código por código no "Campo de Entrada":

  1. Crie a tartaruga com o nome que desejar, neste exemplo, a tartaruga se chamará mat. Assim, tem-se $mat = Tartaruga()$;
  2. Digite $TartarugaIrParaFrente(mat, 1)$;
  3. Digite $TartarugaIrParaEsquerda(mat, pi/2)$;
  4. Digite $TartarugaIrParaFrente(mat, 1)$;
  5. Digite $TartarugaIrParaEsquerda(mat, pi/2)$;
  6. Digite $TartarugaIrParaFrente(mat, 1)$;
  7. Digite $TartarugaIrParaEsquerda(mat, pi/2)$;
  8. Digite $TartarugaIrParaFrente(mat, 1)$;
  9. Digite $TartarugaIrParaEsquerda(mat, pi/2)$.

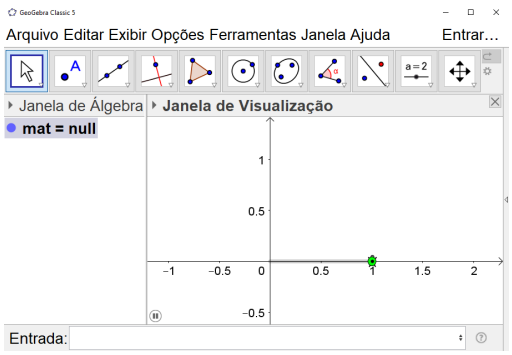
  A Figura 1.9 exibe esse processo.



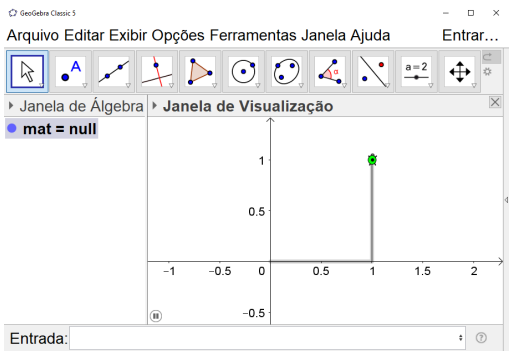
(a) Criação da tartaruga de nome mat (passo 1).



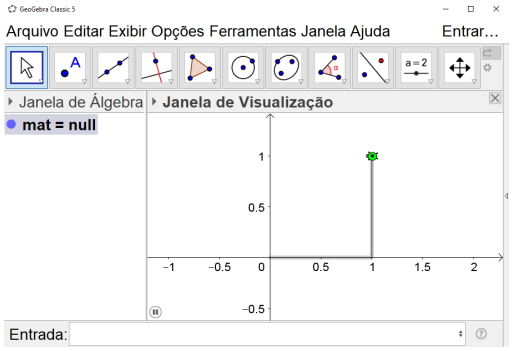
(b) Tartaruga, de nome mat, andando uma unidade para frente (passo 2).



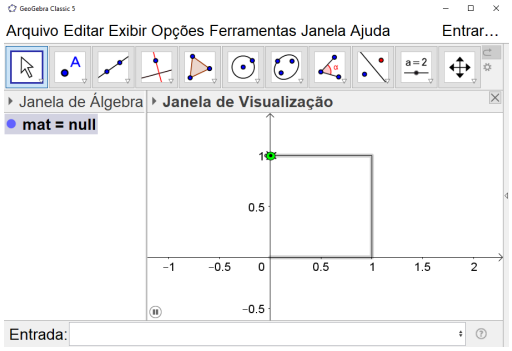
(c) Tartaruga, de nome mat, girando $\frac{\pi}{2}$ no sentido anti-horário. (Passo 3)



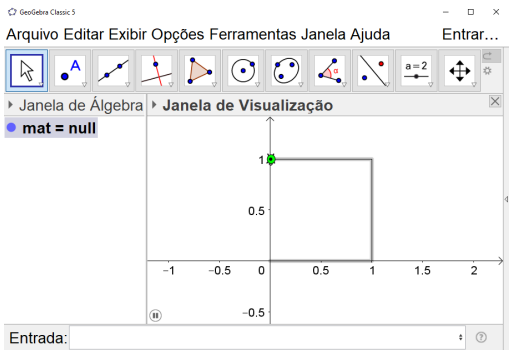
(d) Tartaruga, de nome mat, andando uma unidade para frente (passo 4).



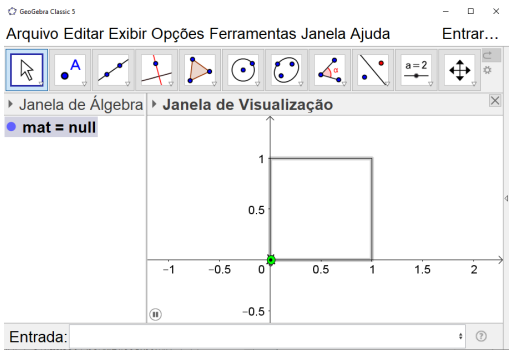
(e) Tartaruga, de nome mat, girando $\frac{\pi}{2}$ no sentido anti-horário (passo 5).



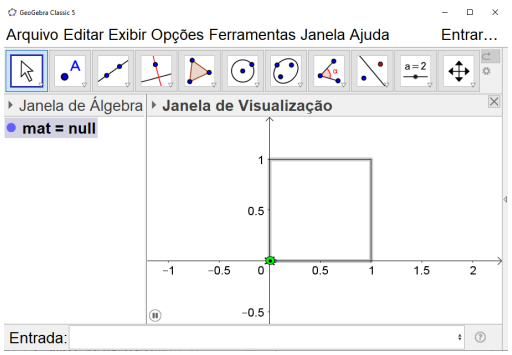
(f) Tartaruga, de nome mat, andando uma unidade para frente (passo 6).



(g) Tartaruga, de nome mat, girando $\frac{\pi}{2}$ no sentido anti-horário (passo 7).



(h) Tartaruga, de nome mat, andando uma unidade para frente (passo 8).



(i) Tartaruga, de nome mat, girando $\frac{\pi}{2}$ no sentido anti-horário (passo 9).

Figura 1.9: Processo de criação de um quadrado utilizando os comandos da tartaruga.

Pode-se perceber que é exaustivo digitar 9 códigos um por um no "Campo de Entrada". Logo, o comando $Repetir$ auxilia na otimização de todo esse processo da seguinte maneira:

Note que os passos 2, 4, 6 e 8 são todos iguais, ou seja, o comando

$$TartarugaIrParaFrente(mat, 1)$$

aparece quatro vezes em todo o processo de criação do quadrado de lado uma unidade.

O mesmo acontece com o comando

$$TartarugaIrParaEsquerda(mat, pi/2)$$

nos passos 3, 5, 7 e 9.

Portanto, podemos definir o comando $Repetir$ como

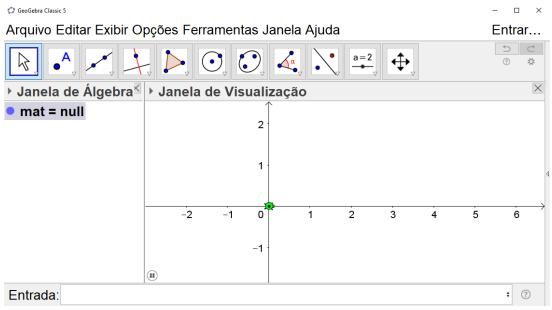
$$Repetir(4, TartarugaIrParaFrente(mat, 1), TartarugaIrParaEsquerda(mat, pi/2)),$$

em que, 4 é o número de vezes que os comandos $TartarugaIrParaFrente(mat, 1)$ e $TartarugaIrParaEsquerda(mat, pi/2)$ devem ser executados, nesta mesma ordem em que foram colocados. Em outras palavras, a tartaruga, de nome mat, se movimenta uma unidade para frente e gira seu corpo em um ângulo de $\frac{\pi}{2}$ radianos no sentido anti-horário, quatro vezes.

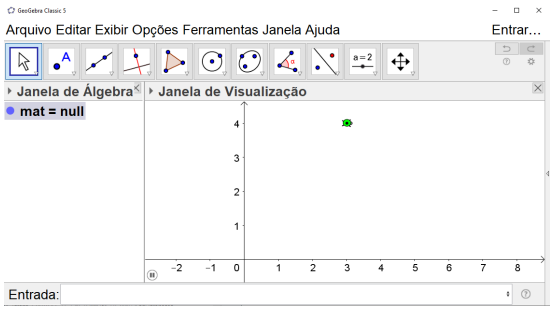
- $DefinirCoordenadas(< Objeto >, < x >, < y >)$:

  Este comando possui a função de definir as próximas coordenadas x e y em $< x >$ e $< y >$, respectivamente, de determinado objeto em $< Objeto >$.

  Por exemplo, utilizando a tartaruga, neste caso, de nome mat e, com ela na origem do plano cartesiano, suponha que o usuário queira transladá-la para as coordenadas $(x, y) = (3, 4)$. Assim, basta colocar mat em $< Objeto >$, 3 em $< x >$ e 4 em $< y >$. A figura 1.10 mostra esse processo.



(a) Tartaruga, de nome mat, na origem do plano cartesiano.



(b) Tartatura, de nome mat, transladada para as coordenadas $(x, y) = (3, 4)$.

Figura 1.10: Tartaruga, de nome mat, transladada da origem do plano cartesiano para as coordenadas $(x, y) = (3, 4)$.

- $DefinirValor(< Objeto >, < Objeto >)$:

Este comando possui a função de redefinir valores de um objeto qualquer.

Por exemplo, suponha que o usuário queira incrementar em uma unidade uma variável $n = 0$ criada previamente a partir do "Campo de Entrada". Para isso, basta colocar n no primeiro $< Objeto >$ e $n + 1$ no segundo $< Objeto >$.

Como exemplo de aplicação do uso da tartaruga com o manual da Seção 1.2.1 mostra-se no Exemplo 1.1 que está relacionado com o conceito de tesselação, cujos detalhes podem ser encontrados no Capítulo 6. Além disso, é mostrado uma interessante propriedade geométrica das tesselações, enunciada e demonstrada no Teorema 1.1.

Exemplo 1.1. Neste exemplo é utilizado o conceito de tesselação (veja Capítulo 6, Seção 6.1 para mais detalhes teóricos). Sucintamente, uma tesselação unicelular regular é representar o plano $\mathbb{R}^2$ como uma união de polígonos congruentes com um polígono regular chamado de gerador da tesselação, os quais tem no máximo um lado em comum. Diz-se que a tesselação é um cobrimento de $\mathbb{R}^2$.

Tem sido demonstrado, em [16], que há somente três tesselações unicelulares regulares em que o polígono gerador é um triângulo equilátero, um quadrado, ou um hexágono regular. Além disso, Grünbaum e Shephard [16] denotam esses três tipos de tesselações como $(3, 3, 3, 3, 3, 3) = (3^6)$, $(4, 4, 4, 4) = (4^4)$ e $(6, 6, 6) = (6^3)$, respectivamente. A notação (3^6) indica que o polígono regular tem 3 lados e que, a partir de qualquer vértice desse polígono, partem outros 5 polígonos regulares da mesma forma e do mesmo tamanho. Ou seja, dado um triângulo equilátero qualquer, a partir de um de seus vértices, partem outros 5 triângulos equiláteros. Analogamente para as notações (4^4) e (6^3).

Assim, demonstra-se o Teorema 1.1.

Teorema 1.1. *As únicas tesselações unicelulares regulares são as que possuem, como polígono gerador, um triângulo equilátero, um quadrado ou um hexágono regular.*

Demonstração. Seja K o número de polígonos regulares congruentes ao redor de um vértice comum destes polígonos, denominado nó. Sabe-se, da Geometria Euclidiana [13] que o valor do ângulo interno α_i de um polígono regular de n lados é dado por:

$$\alpha_i = \frac{(n-2)\pi}{n}.$$

Note que para acontecer o cobrimento do plano com polígonos regulares, a soma de todos os ângulos internos ao redor de um nó deve ser 2π. Sabe-se, pela hipótese do teorema que, para haver o cobrimento do plano, não se deve deixar espaços em branco entre os polígonos ou ficarem sobrepostos, fato que limita o número de polígonos que atendam a esse critério.

Dessa forma, tem-se $K\alpha_i = 2\pi$. Assim, $K = \dfrac{2\pi}{\alpha_i}$ e, portanto,

$$K = \frac{2n}{(n-2)}. \tag{1.1}$$

Para que (1.1) seja verdadeira, deve-se ter $n \geq 3$, pois não existem polígonos de 1 ou 2 lados. Também tem-se que $K \geq 3$, pois não faz sentido colocar um ou dois polígonos ao redor de um nó.

Assim, do fato que $K \geq 3$, então $\dfrac{2n}{(n-2)} \geq 3$, ou seja, $2n \geq 3(n-2)$ e, portanto, $n \leq 6$.

Logo, tem-se o seguinte intervalo para n:

$$3 \leq n \leq 6,\ n \in \mathbb{Z}. \tag{1.2}$$

Agora, analisa-se a equação (1.1) a partir do intervalo (1.2).

Caso 1) $n = 3$:

$$K = \frac{2 \cdot 3}{(3-2)} = 6.$$

Então, são 6 polígonos regulares de 3 lados cada, ou seja, 6 triângulos equiláteros ao redor de um nó, sem espaços e sem sobreposições.

Caso 2) $n = 4$:

$$K = \frac{2 \cdot 4}{(4-2)} = 4.$$

Então, são 4 polígonos regulares de 4 lados cada, ou seja, 4 quadrados ao redor de um nó, sem espaços e sem sobreposições.

Caso 3) $n = 5$:

$$K = \frac{2 \cdot 5}{(5-2)} = \frac{10}{3}.$$

Como $K \in \mathbb{Z}$, então não pode-se dispor pentágonos ao redor de um nó sem haver espaços e sobreposições.

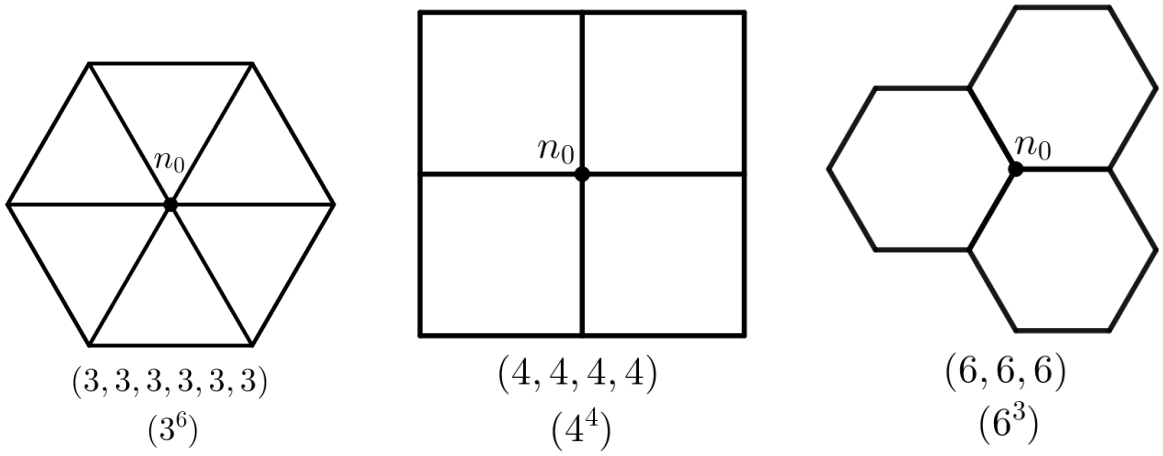
Caso 4) $n = 6$:

$$K = \frac{2 \cdot 6}{(6-2)} = 3.$$

Então, são 3 polígonos regulares de 6 lados cada, ou seja, 3 hexágonos regulares ao redor de um nó, sem espaços e sem sobreposições.

Portanto, há somente três tesselações unicelulares regulares geradas por: triângulos equiláteros, quadrados ou hexágonos regulares. ∎

A Figura 1.11 mostra os polígonos regulares ao redor do nó n_0, com suas respectivas notações.



(a) Nó n_0 rodeado por 6 triângulos equiláteros sem haver espaços e sobreposições.

(b) Nó n_0 rodeado por 4 quadrados sem haver espaços e sobreposições.

(c) Nó n_0 rodeado por 3 hexágonos regulares sem haver espaços e sobreposições.

Figura 1.11: Nó n_0 rodeado de polígonos regulares sem haver espaços e sobreposições.

O Teorema 1.1 serviu como motivação para a criação de tesselações unicelulares regulares com a tartaruga do GeoGebra. A seguir serão apresentadas dinâmicas para a criação dessas tesselações, que temos demonstrado que seus polígonos geradores são triângulos equiláteros, quadrados ou hexágonos regulares. Em todos esses casos, procede-se a construir as tesselações para uma modelagem discreta de pavimentações 15×15 e, considerando o lado dos polígonos regulares $l = 1$, a fim de tornar as dinâmicas mais simples e práticas.

1.2.2 Tesselação Formada por Triângulos Equiláteros

O processo para criação de tesselações deste tipo é bem simples, porém exigem conhecimento sobre geometria básica de triângulos equiláteros. A dinâmica é apresentada no seguinte passo a passo:

1. Criar dois contadores $n = 0$ e $m = 0$ no "Campo de Entrada" do GeoGebra, que servirão para redefinir as coordenadas da tartaruga.

2. Criar a tartaruga, neste caso, de nome mat, ou seja $mat = Tartaruga()$.

3. Digitar o seguinte código no "Campo de Entrada":

$$\begin{aligned} &Repetir(15, Repetir(2, Repetir(15, Repetir(3, TartarugaIrParaFrente(mat, 1),\\ &TartarugaIrParaEsquerda(mat, 2\pi/3)), TartarugaIrParaFrente(mat, 1)),\\ &TartarugaIrParaEsquerda(mat, \pi/3), TartarugaIrParaFrente(mat, 1),\\ &TartarugaIrParaEsquerda(mat, 2\pi/3)),\\ &DefinirCoordenadas(mat, n + 0.5, m + 1 * sqrt(3)/2),\\ &DefinirValor(n, n + 0.5), DefinirValor(m, m + 1 * sqrt(3)/2)). \end{aligned} \tag{1.3}$$

4. Por fim, clica-se no botão de "*play*"  para animar a tartaruga.

Pode-se perceber que na programação do item 3 utilizou-se 4 comandos $Repetir$, um no interior do outro.

Iniciando do primeiro $Repetir$ de dentro para fora, ou seja, referente ao código

$$Repetir(3, TartarugaIrParaFrente(mat, 1), TartarugaIrParaEsquerda(mat, 2\pi/3)),$$

tem a função de fazer a tartaruga andar uma unidade para frente e girar no sentido anti-horário um ângulo de $\frac{2\pi}{3}$ radianos, repetindo esses movimentos, nessa mesma ordem, 3 vezes. Isto resulta em um triângulo equilátero de lado $l = 1$. Veja Figura 1.12.

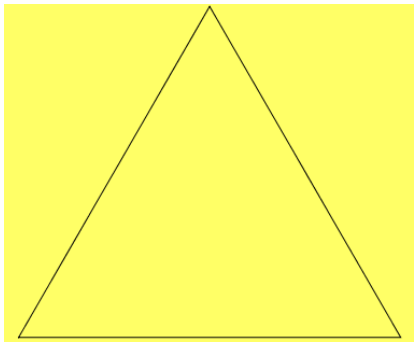


Figura 1.12: Triângulo Equilátero de lado $l = 1$.

O segundo $Repetir$ de dentro para fora do código (1.3), ou seja, referente à parcela

$$Repetir(15, Repetir(3, TartarugaIrParaFrente(mat, 1),$$
$$TartarugaIrParaEsquerda(mat, 2\pi/3)), TartarugaIrParaFrente(mat, 1)),$$

tem a função de fazer a tartaruga desenhar, 15 vezes, um triângulo equilátero e, além disso, após desenhar cada triângulo, ela anda uma unidade para frente, com o objetivo de se criar o próximo triângulo, e assim sucessivamente até criar a primeira fileira horizontal composta por 15 triângulos equiláteros voltados para cima. Veja Figura 1.13.

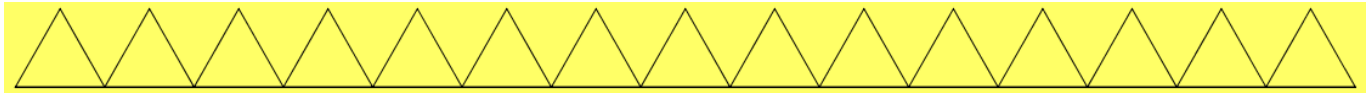


Figura 1.13: Primeira fileira horizontal composta por 15 triângulos equiláteros de lado $l = 1$ voltados para cima.

O terceiro $Repetir$ de dentro para fora do código (1.3), ou seja, referente à:

$$Repetir(2, Repetir(15, Repetir(3, TartarugaIrParaFrente(mat, 1),$$
$$TartarugaIrParaEsquerda(mat, 2\pi/3)), TartarugaIrParaFrente(mat, 1)),$$
$$TartarugaIrParaEsquerda(mat, \pi/3), TartarugaIrParaFrente(mat, 1),$$
$$TartarugaIrParaEsquerda(mat, 2\pi/3)),$$

tem a função de fazer a tartaruga, após traçar toda a fileira horizontal de triângulos equiláteros, traçar a fileira consecutiva destes mesmos triângulos. E, neste caso, eles se encontram voltados para baixo. Veja Figura 1.14.

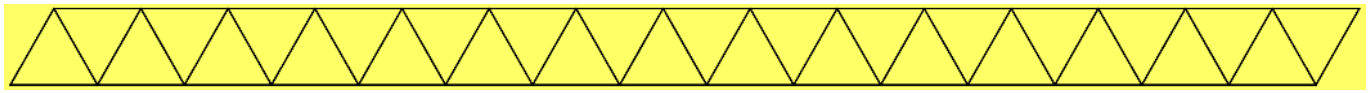


Figura 1.14: Primeira fileira horizontal composta por 15 triângulos equiláteros de lado $l = 1$ voltados para cima, em conjunto com, a fileira consecutiva de triângulos voltados para baixo.

Por último, o quarto $Repetir$ de dentro para fora do código (1.3), na sua totalidade, tem a função de fazer a tartaruga realizar todos os outros 3 $Repetir$, 15 vezes, fazendo com que a tartaruga desenhe quinze fileiras verticais de triângulos equiláteros.

Além disso, a cada duas fileiras horizontais de triângulos, uma voltada para cima e a outra voltada para baixo, respectivamente, as coordenadas da tartaruga são redefinidas para que ela retorne ao vértice superior do primeiro triângulo equilátero, da última fileira modelada de triângulos voltados para cima. Para melhor visualização, veja Figura 1.15, cujo vértice superior está representado pelo ponto C, onde ela retorna com a finalidade de que realize as próximas duas fileiras horizontais e assim sucessivamente.

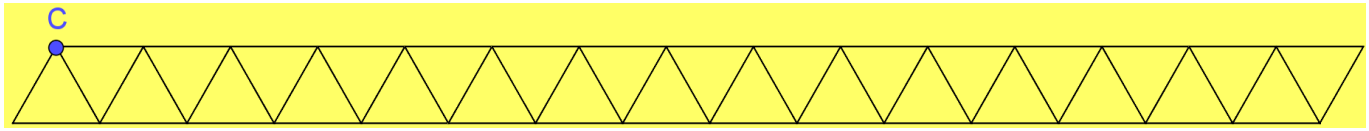


Figura 1.15: Ponto C de retorno da tartaruga para realização da próxima fileira de triângulos equiláteros voltados para cima e para baixo.

Isto pode ser feito com o auxílio dos contadores iniciais n e m a partir do código

$$DefinirCoordenadas(mat, n + 0.5, m + 1 * sqrt(3)/2),$$

cujas coordenadas x da tartaruga é redefinida como $n + 0.5$, ou seja, ela anda metade do passo original (metade do lado do triângulo equilátero de lado $l = 1$) e a coordenada y é redefinida como $m+1*sqrt(3)/2$, ou seja, ela sobe a altura de um triângulo equilátero de lado $l = 1$. Veja Figura 1.16.

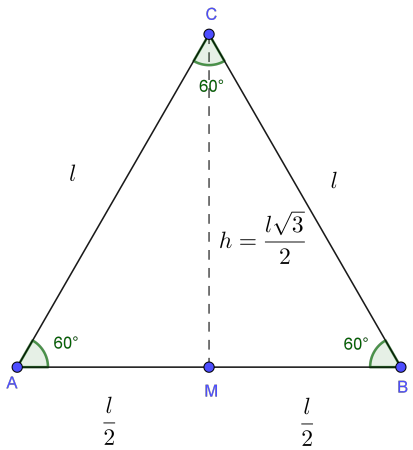


Figura 1.16: Triângulo Equilátero de lado l e altura $h = \dfrac{l\sqrt{3}}{2}$.

Estes contadores também são incrementados, a partir dos códigos

$$DefinirValor(n, n + 0.5) \quad \text{e} \quad DefinirValor(m, m + 1 * sqrt(3)/2),$$

em que $n = n + 0.5$, ou seja, o contador n, a cada vez que termina toda a programação do último repetir, é incrementado em 0.5 unidades. O mesmo processo acontece com o contador m em que:

$$m = m + 1 * sqrt(3)/2,$$

ou seja, é incrementado em $\dfrac{\sqrt{3}}{2}$ unidades, que representa o valor da altura do triângulo equilátero de lado $l = 1$. O resultado da programação pode ser visualizado na Figura 1.17.

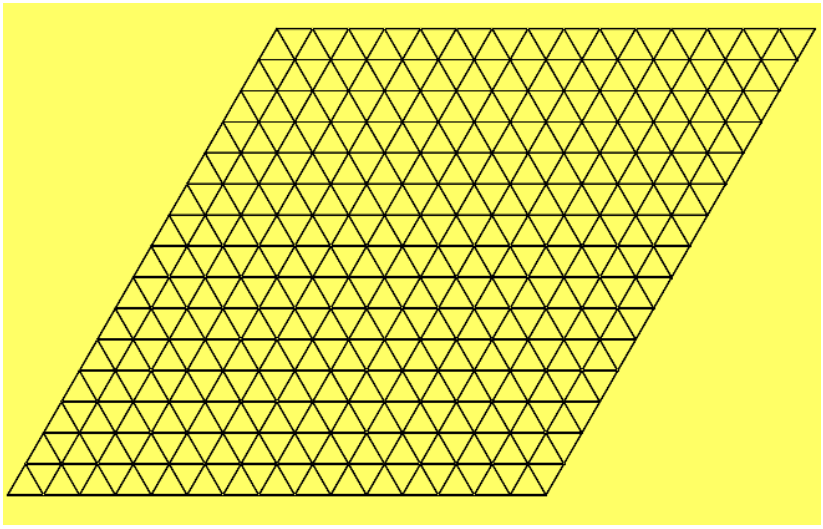


Figura 1.17: Tesselação Unicelular Regular formada por Triângulos Equiláteros.

1.2.3 Tesselação Formada por Quadrados

O processo para criação de tesselações deste tipo é bem mais simples que as tesselações feitas por triângulos equiláteros ou hexágonos regulares. A dinâmica é apresentada no seguinte passo a passo:

1. Criar um contador $n = 0$ no "Campo de Entrada" do GeoGebra, que servirá para redefinir as coordenadas da tartaruga.

2. Criar a tartaruga, neste caso, de nome mat, ou seja $mat = Tartaruga()$.

3. Digitar o seguinte código no "Campo de Entrada":

$$\begin{aligned} &Repetir(15, Repetir(15, Repetir(4, TartarugaIrParaFrente(mat, 1), \\ &TartarugaIrParaEsquerda(mat, \pi/2)), TartarugaIrParaFrente(mat, 1)), \\ &DefinirValor(n, n + 1), DefinirCoordenadas(mat, 0, n)). \end{aligned} \tag{1.4}$$

4. Por fim, clica-se no botão de "*play*"  para animar a tartaruga.

Pode-se perceber que na programação do item 3 utilizou-se 3 códigos $Repetir$, um no interior do outro. Iniciando do primeiro $Repetir$ de dentro para fora, ou seja, referente ao código

$$Repetir(4, TartarugaIrParaFrente(mat, 1), TartarugaIrParaEsquerda(mat, \pi/2)),$$

tem a função de fazer a tartaruga andar uma unidade para frente e girar no sentido anti-horário $\frac{\pi}{2}$ radianos, repetindo esses movimentos, nessa mesma ordem, 4 vezes. Isto resulta em um quadrado de lado $l = 1$ (Figura 1.18).

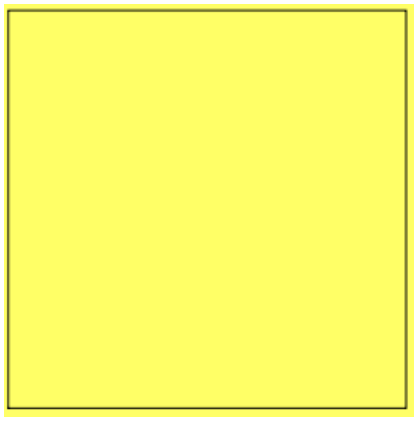


Figura 1.18: Quadrado de lado $l = 1$.

O segundo $Repetir$ de dentro para fora do código (1.4), ou seja, referente à parcela

$$Repetir(15, Repetir(4, TartarugaIrParaFrente(mat, 1),$$
$$TartarugaIrParaEsquerda(mat, \pi/2)), TartarugaIrParaFrente(mat, 1)),$$

tem a função de fazer a tartaruga desenhar, 15 vezes, um quadrado e, além disso, após desenhar cada quadrado, ela anda uma unidade para frente, com o objetivo de se criar o próximo quadrado, e assim sucessivamente até criar a primeira fileira horizontal composta por 15 quadrados de lado $l = 1$. Veja Figura 1.19.

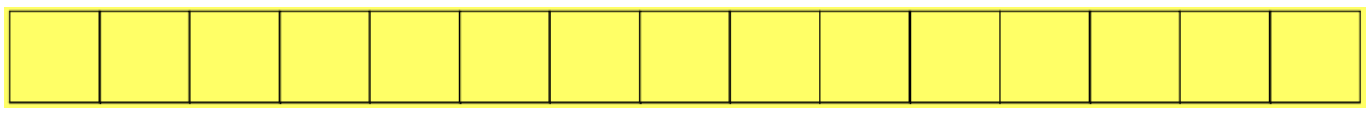


Figura 1.19: Primeira fileira horizontal composta por 15 quadrados lado $l = 1$.

Por último, o terceiro $Repetir$ de dentro para fora do código (1.4), na sua totalidade, tem a função de fazer a tartaruga realizar todos os outros 2 comandos $Repetir$, 15 vezes, fazendo com que a ela desenhe quinze fileiras verticais de quadrados.

Além disso, a cada fileira horizontal de quadrados, as coordenadas da tartaruga são redefinidas para que ela realize a próxima fileira horizontal consecutiva com o auxílio do contador inicial n, a partir do código

$$DefinirCoordenadas(mat, 0, n),$$

em que, as coordenadas x e y da tartaruga são redefinidas como 0 e n, respectivamente, ou seja, ela retorna ao eixo y, porém com uma altura n.

Este contador também é incrementado, a partir do código

$$DefinirValor(n, n + 1),$$

em que $n = n + 1$, ou seja, o contador n, a cada vez que termina toda a programação do último repetir, antes de definir as próximas coordenadas da tartaruga, é incrementado em 1 unidade.

O resultado da programação pode ser visualizado na Figura 1.20.

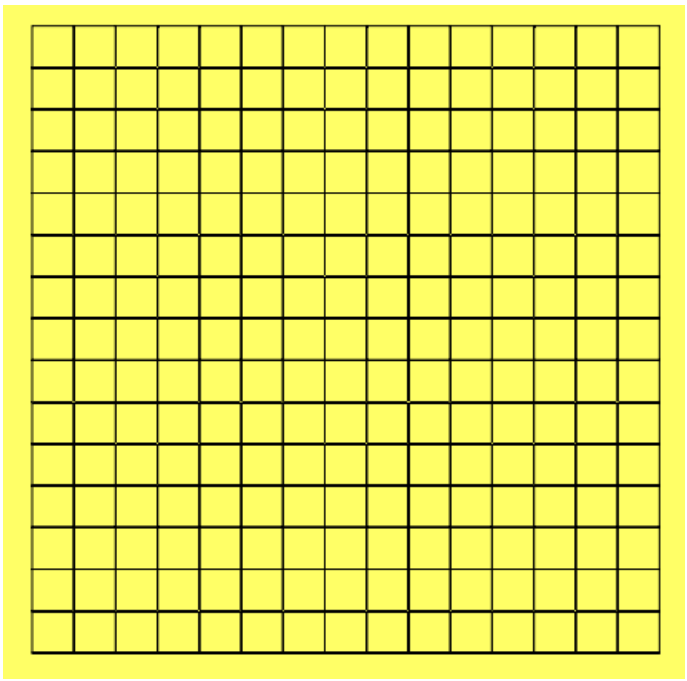


Figura 1.20: Tesselação Unicelular Regular formada por Quadrados.

1.2.4 Tesselação Formada por Hexágonos Regulares

O processo para criação de tesselações deste tipo é o mais complexo dentre as unicelulares regulares, o qual exige conhecimento intermediário sobre geometria básica de triângulos. A dinâmica é apresentada no seguinte passo a passo:

1. Criar dois contadores $n = 0$ e $m = 0$ no "Campo de Entrada" do GeoGebra, que servirão para redefinir as coordenadas da tartaruga.

2. Criar a tartaruga, neste caso, de nome mat, ou seja $mat = Tartaruga()$.

3. Digitar o seguinte código no "Campo de Entrada": $TartarugaIrParaDireita(mat, \pi/6)$, cuja finalidade é de iniciar a tartaruga com um giro de $\dfrac{\pi}{6}$ radianos no sentido horário.

4. Digitar o seguinte código no "Campo de Entrada":

$$\begin{aligned} &Repetir(15, Repetir(15, Repetir(7, TartarugaIrParaFrente(mat, 1),\\ &TartarugaIrParaEsquerda(mat, \pi/3)), TartarugaIrParaFrente(mat, 1),\\ &TartarugaIrParaDireita(mat, \pi/3)), DefinirValor(m, m + 1.5),\\ &DefinirValor(n, n + (1 * sqrt(3)/2)), DefinirCoordenadas(mat, n, m)). \end{aligned} \tag{1.5}$$

5. Por fim, clica-se no botão de "*play*"  para animar a tartaruga.

Pode-se perceber que na programação do item 4 foram utilizados 3 comandos $Repetir$, um no interior do outro.

Iniciando do primeiro $Repetir$ de dentro para fora, ou seja, referente ao código

$$Repetir(7, TartarugaIrParaFrente(mat, 1), TartarugaIrParaEsquerda(mat, \pi/3)),$$

tem a função de fazer a tartaruga andar uma unidade para frente e girar seu corpo no sentido anti-horário um ângulo de $\frac{\pi}{3}$ radianos, repetindo esses movimentos, nessa mesma ordem, 7 vezes. Isto resulta em um hexágono regular de lado $l = 1$. Veja Figura 1.21.

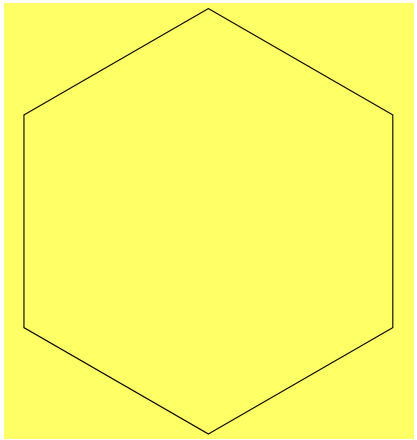


Figura 1.21: Hexágono Regular de lado $l = 1$.

O segundo $Repetir$ de dentro para fora do código (1.5), ou seja, referente à parcela

$$\begin{aligned} &Repetir(15, Repetir(7, TartarugaIrParaFrente(mat, 1),\\ &TartarugaIrParaEsquerda(mat, \pi/3)), TartarugaIrParaFrente(mat, 1),\\ &TartarugaIrParaDireita(mat, \pi/3)), \end{aligned}$$

tem a função de fazer a tartaruga desenhar, 15 vezes, um hexágono regular, com a finalidade de criar uma fileira horizontal composta por 15 hexágonos regulares. Veja Figura 1.22.

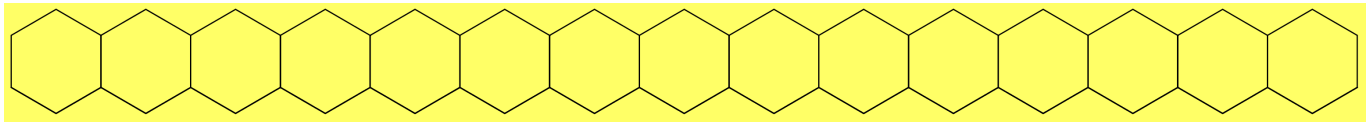


Figura 1.22: Primeira fileira horizontal composta por 15 hexágonos regulares de lado $l = 1$.

Por último, o terceiro $Repetir$ de dentro para fora do código (1.5), na sua totalidade, tem a função de fazer a tartaruga realizar todos os outros 3 $Repetir$, 15 vezes, fazendo com que a tartaruga desenhe quinze fileiras verticais de hexágonos regulares.

Para que a tartaruga realize a segunda fileira horizontal de hexágonos, é preciso que ela retorne ao ponto A da Figura 1.23.

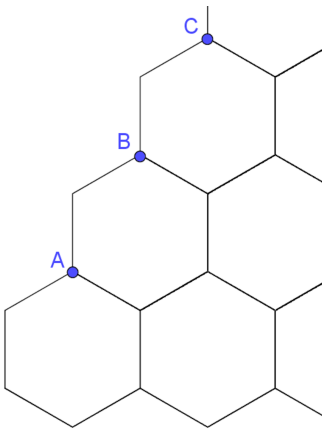


Figura 1.23: Pontos A, B e C de retorno da tartaruga após a realização das primeira, segunda e terceira fileiras horizontais respectivamente.

Para determinar as coordenadas do ponto A deve-se utilizar algumas ferramentas da geometria euclidiana [13]. A fim de tornar a explicação subsequente mais ilustrativa, veja Figura 1.24.

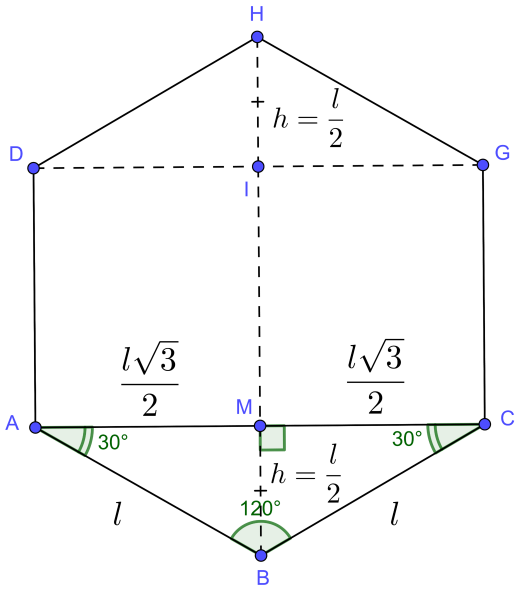


Figura 1.24: Hexágono Regular de lado l.

Supondo que o ponto A da Figura 1.24 esteja na origem do plano cartesiano, determina-se as coordenadas do ponto H da seguinte maneira:

1. Utiliza-se a Lei dos Senos [13] no triângulo isóceles $\triangle$ABC para calcular o lado AC.

2. Tem-se que a altura e a mediana de um triângulo isóceles são coincidentes, ou seja, ao traçar a altura BM ou h do triângulo $\triangle$ABC, tem-se que M é o ponto médio do lado AC, ou seja, AM $=$ MC $= \dfrac{l\sqrt{3}}{2}$. E, utilizando o Teorema de Pitágoras [13] no triângulo $\triangle$BMC tem-se que $h = \dfrac{l}{2}$.

3. Pela congruência dos triângulos $\triangle$ABC e $\triangle$DHG devido à igualdade de dois lados e o ângulo compreendido entre eles (caso LAL [13]), tem-se que HI $= h = \dfrac{l}{2}$ e DI $=$ IG $= \dfrac{l\sqrt{3}}{2}$. Assim, as coordenadas de H são $\left(\dfrac{l\sqrt{3}}{2}, \dfrac{3l}{2}\right)$.

A seguir, sabendo que foi escolhido $l = 1$, utiliza-se dois comandos para incrementar os contadores n e m, ou seja

$$DefinirValor(n, n + (1 * sqrt(3)/2)) \quad \text{e} \quad DefinirValor(m, m + 1.5),$$

os quais, representam, respectivamente, as próximas coordenadas x e y para onde a tartaruga deverá transladar-se.

Por fim, com o auxílio destes dois contadores n e m, redefine-se as coordenadas da tartaruga para que ela se desloque até o ponto A da Figura 1.23 e realize a segunda fileira horizontal de hexágonos regulares, utilizando o código

$$DefinirCoordenadas(mat, n, m),$$

em que, n é a coordenada x e m é a coordenada y.

Note que, o comando $DefinirCoordenadas(mat, n, m)$ translada a tartaruga para pontos distintos a cada vez que o último comando $Repetir$, de dentro para fora do código (1.5), é realizado. Ou seja, após a tartaruga desenhar a primeira linha de hexágonos, ela translada para o ponto A. Após desenhar a segunda linha, ela translada para o ponto B da Figura 1.23 e, assim, sucessivamente.

O resultado da programação pode ser visualizado na Figura 1.25.

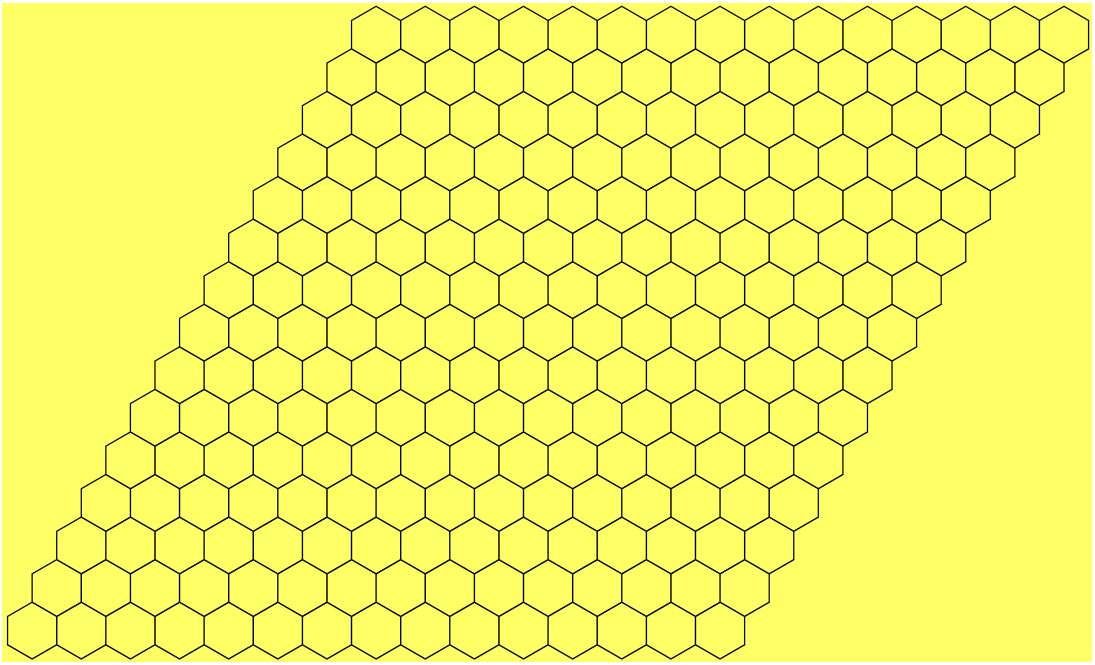


Figura 1.25: Tesselação Unicelular Regular formada por Hexágonos Regulares.

Desta forma, tem-se apresentado exemplos da utilização da tartaruga explicitados nas tesselações unicelulares regulares, como contribuição para leitores interessados na utilização desta ferramenta do GeoGebra.

Outra ferramenta utilizada nesta pesquisa, como já mencionado anteriormente, é a janela de programação avançada o que inclui a linguagem *JavaScript* que é o assunto principal da Seção 1.3.

1.3 *JavaScript*

A fim de competir com a *Microsoft*, pela adoção de tecnologias e plataformas da *web*, a *Netscape Communications* colaborou com a *Sun Microsystems* para criar uma linguagem de *script*, mais conhecida como *JavaScript*. Para desempenhar este papel e cumprir este propósito, em 1995, a *Netscape Communications* recrutou o cientista da computação e, também matemático, Brendan Eich. O processo de incorporação da linguagem é descrita pelo próprio Brendan como é mostrado neste trecho de uma entrevista em [28]:

> [...] Foi tudo dentro de seis meses, de maio a dezembro (1995), que se tornou Mocha e, em seguida, LiveScript. E então, no início de dezembro, a Netscape e a Sun fizeram um acordo de licença e ela se tornou JavaScript. E a ideia era torná-la uma linguagem de script complementar para acompanhar o Java, com a linguagem compilada.

Após o surgimento do *JavaScript*, os trabalhos dos desenvolvedores de páginas e sistemas *web* ficaram muito mais flexíveis, pois esta linguagem de *scripting* possui a habilidade de executar um código direto do navegador, sem que ele seja compilado. Por conta disso, o *JavaScript* é conhecido como uma linguagem *client-side*, em que não há a necessidade do processamento do servidor para determinada tarefa e sim apenas do usuário, ou no caso, de seu próprio navegador.

O *Java* é uma linguagem orientada à criação de objetos e que independe de plataformas para executar seus códigos. Diferente do *JavaScript*, cuja plataforma é o navegador. Outra característica que difere as duas linguagens é que, de acordo com Goodman et al. [15]

> Ao invés do vocabulário completo da linguagem de programação do Java (e da abordagem orientada a objetos conceitualmente mais difícil de aprender), o JavaScript tinha um vocabulário pequeno e um modelo de programação mais facilmente digestível.

Segundo Goodman et al. [15], a *Netscape* adotou o nome de *JavaScript* por ter boas razões de *marketing*, justamente pelo fato que a linguagem de programação *Java* estava na moda na época. Entretanto, Goodman et al. [15] também apontam que muitos dos elementos de sintaxe do *JavaScript* lembram a linguagem *Java*.

Pelo fato do *JavaScript* ser utilizado em páginas *web* e, com isso, não precisar de um compilador para executar seus comandos, tornou-se possível incorporá-lo no GeoGebra , principalmente na criação de *Applets*. Os *Applets* são pequenos programas incorporados nas páginas *web* que foram criados para fornecer recursos interativos para aplicativos que não podiam ser fornecidos apenas pelo *html*.

O GeoGebra suporta duas linguagens de *script*: GeoGebra *Script* (*GGBScript*) e o *JavaScript*. Como descrito no portal do GeoGebra [22]:

> JavaScript é uma linguagem de programação usada por muitas tecnologias da internet. Ao contrário do GeoGebra Script, em JavaScript os comandos não precisam ser executados como uma sequência simples, mas um fluxo de controle (if, while, for) pode ser usado. [...] No GeoGebra, podem-se usar métodos JavaScript especiais que permitem alterar a construção. Esses métodos pertencem ao conjunto de comandos ggbApplet, o que significa que o usuário chama os objetos da forma ggbApplet.methodname(parâmetro,...,parâmetro).

O uso da tartaruga do GeoGebra, juntamente, com o *JavaScript*, traz como vantagem o poder da codificação avançada que permite a criação de dinâmicas mais complexas como as das propostas neste estudo.

Com o objetivo de mostrar a linguagem *JavaScript* do GeoGebra e suas vantagens, fornece-se o mesmo exemplo da tesselação unicelular gerada por um triângulo equilátero, vista na Seção 1.2.2.

Desta forma, escolheu-se criar botões programáveis de nomes “Criar tartaruga”, “Deletar tartaruga” e “Gerar” no GeoGebra que auxilia na execução dos comandos *JavaScript* pela tartaruga. Ver figura 1.26.



(a) Botão “Criar tartaruga”. (b) Botão “Deletar tartaruga”. (c) Botão “Gerar”.

Figura 1.26: Botões programáveis criados para auxiliar a execução dos comandos *JavaScript* pela tartaruga.

Cada botão possui comandos específicos que, ao clicá-los, a tartaruga desempenhará determinada função. Para determinar qual função a tartaruga deve realizar, basta clicar com o botão direito do *mouse* em cima de algum botão e clicar em "Propriedades". Abrirá uma janela na lateral direita do GeoGebra como mostra na Figura 1.27.

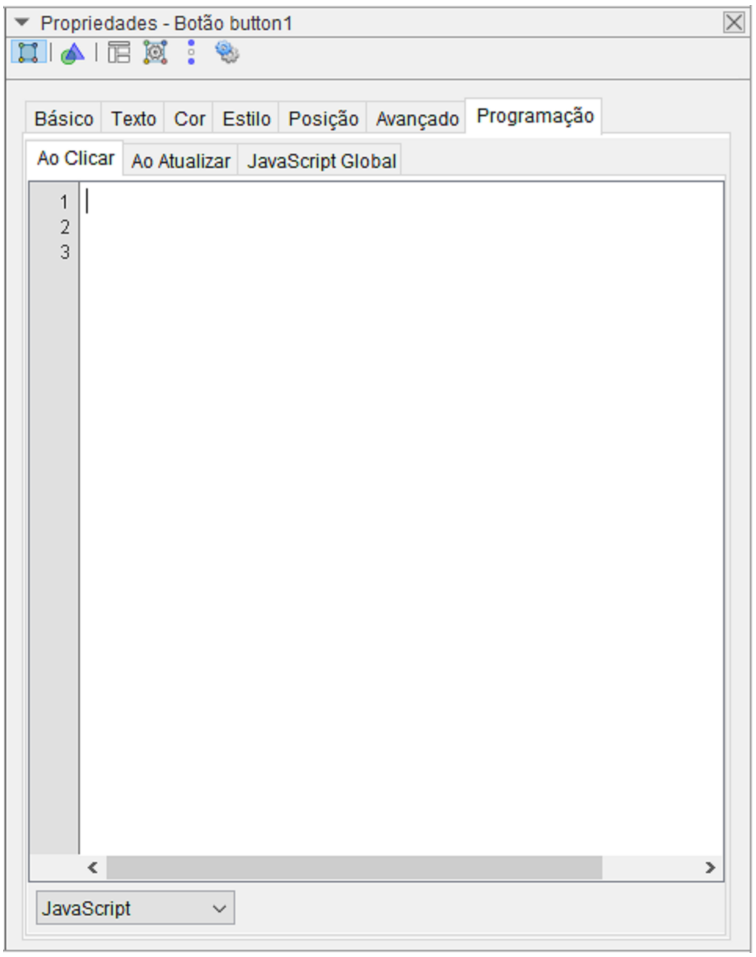


Figura 1.27: Janela de Propriedades de um botão criado previamente.

Pode-se perceber que existem três abas: "Ao Clicar", "Ao Atualizar" e "JavaScript Global". Em todas as modelagens desenvolvidas neste trabalho, foram utilizadas apenas as abas "Ao Clicar" e "JavaScript Global".

Na aba "Ao Clicar", encontram-se os comandos que a tartaruga realizará. Assim, para esta modelagem, são inseridos os seguintes códigos específicos determinados para cada botão:

- Para o botão $1.26(a)$, foram inseridos os comandos:

$$posicionar(); \tag{1.6}$$

$$ggbApplet.evalCommand(\text{“}n = 0\text{”}); \tag{1.7}$$

$$ggbApplet.evalCommand(\text{“}m = 0\text{”}); \tag{1.8}$$

O comando $posicionar()$ chama uma função definida na aba "JavaScript Global".

Os comandos $ggbApplet.evalCommand(\text{“}n = 0\text{”}))$ e $ggbApplet.evalCommand(\text{“}m = 0\text{”})$ criam duas variáveis n e m que auxiliarão na definição de novas coordenadas da tartaruga posteriormente.

- Para o botão $1.26(b)$, foram inseridos os comandos:

$$deletarTartaruga(); \tag{1.9}$$
$$ggbApplet.evalCommand(\text{“}Delete[n]\text{”}); \tag{1.10}$$
$$ggbApplet.evalCommand(\text{“}Delete[m]\text{”}); \tag{1.11}$$

O comando $deletarTartaruga()$ chama uma função definida na aba “JavaScript Global”.

Os comandos $ggbApplet.evalCommand(\text{“}Delete[n]\text{”})$ e $ggbApplet.evalCommand(\text{“}Delete[m]\text{”})$ deletam as duas variáveis criadas n e m.

- Para o botão $1.26(c)$, foi inserido o comando:

$$turtle(); \tag{1.12}$$

O comando $turtle()$ chama uma função definida na aba “JavaScript Global”.

Na aba “JavaScript Global”, é inserida toda a programação que realizará a modelagem, tanto os códigos que se encontram nos botões, quanto os que movimentarão a tartaruga. Neste exemplo, o código usado para modelar a tesselação unicelular gerada por um triângulo equilátero é mostrado a seguir:

```
function ggbOnInit() {}

function posicionar() {
    ggbApplet.evalCommand("mat=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[]");
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[mat]");
}

function turtle() {
    for (var i = 0; i < 15; i++) {
        for (var j = 0; j < 2; j++) {
            for (var k = 0; k < 15; k++) {
```

```
                for (var l = 0; l < 3; l++) {
                    ggbApplet.evalCommand("TurtleForward[mat, 1]");
                    ggbApplet.evalCommand("TurtleLeft[mat, 2*pi/3]");
                }
                ggbApplet.evalCommand("TurtleForward[mat, 1]");
            }
            ggbApplet.evalCommand("TurtleLeft[mat, pi/3]");
            ggbApplet.evalCommand("TurtleForward[mat, 1]");
            ggbApplet.evalCommand("TurtleLeft[mat, 2*pi/3]");
        }
        ggbApplet.evalCommand("SetCoords[mat,n+0.5,m+(1*sqrt(3)/2)]");
        ggbApplet.evalCommand("n=n+0.5");
        ggbApplet.evalCommand("m=m+(1*sqrt(3)/2)");
    }
}
```

Percebe-se que o código acima foi organizado em 4 funções, $ggbOnInit()$, $posicionar()$, $deletarTartaruga()$, $turtle()$ e, cada função dessa retorna uma ação diferente que a tartaruga realiza. A primeira função definida como $ggbOnInit()$, inicia todo o código da aba "JavaScript Global". A segunda função definida, de nome $posicionar()$, determina a criação de uma tartaruga de nome mat e inicia sua animação. Além disso, esta função é realizada no momento em que se clica no botão "Criar tartaruga", pois é o local que foi estabelecido para retornar a ação desta função $posicionar()$.

O mesmo acontece com a terceira função, ou seja, $deletarTartaruga()$, que, neste caso, deleta a tartaruga de nome mat e só se realiza ao clicar no botão "Deletar tartaruga".

Por fim, a função $turtle()$ que movimenta a tartaruga no plano cartesiano para modelar a tesselação desejada. Note que, foi utilizado o código $for(...)$, quatro vezes, um no interior do outro, e todos dentro da função $turtle()$. O objetivo de se criar um laço de comandos da mesma forma que foi criado em 1.3, desempenha o mesmo papel e na mesma ordem que os quatro $Repetir$. E, da mesma forma que as três funções anteriores, esta só se realiza ao clicar no botão "Gerar".

Observe que há uma pequena diferença na linguagem entre os códigos utilizados em 1.3 dentro dos comandos $Repetir$, com os comandos utilizados dentro dos códigos $for(...)$, ou seja, ao utilizar o *JavaScript* para qualquer modelagem, seus códigos devem ser escritos, necessariamente, em inglês.

Além disso, após a tartaruga realizar os três primeiros comandos $for(...)$, suas coordenadas serão alteradas com o auxílio das variáveis n e m criadas no botão "Criar tartaruga", além de que elas serão incre-

mentadas de acordo com o que foi explicitado na Seção 1.2.2, através dos comandos:

$$ggbApplet.evalCommand("SetCoords[mat, n + 0.5, m + (1 * sqrt(3)/2)]");$$
$$ggbApplet.evalCommand("n = n + 0.5");$$
$$ggbApplet.evalCommand("m = m + (1 * sqrt(3)/2)");$$

A Tabela 1.1 exibe todos os métodos especiais do *JavaScript*, os quais pertencem ao grupo de comandos *ggbApplet*, apresentados na exemplificação do programa, em *JavaScript*, da tesselação proposta, utilizados para todas as modelagens deste trabalho, na aba "JavaScript Global".

Tabela 1.1: Métodos especiais do *JavaScript*, os quais pertencem ao conjunto de comandos *ggbApplet* utilizados no GeoGebra para trabalhar com a tartaruga.

Métodos especiais do *JavaScript* com *ggbApplet*	**Descrição**
boolean evalCommand(String cmdString)	Avalia a *string* fornecida da mesma forma que seria avaliada quando inserida na barra de entrada do GeoGebra. Retorna se a avaliação do comando foi bem-sucedida.
void setTextValue(String objName, String value)	Define o valor do texto do objeto com o nome fornecido. Para qualquer outro tipo de objeto, nada é feito.
void setCoords(String objName, double x, double y)	Define as coordenadas do objeto com o nome fornecido. Nota: se o objeto especificado não for um ponto, vetor, linha ou objeto absolutamente posicionado (texto, botão, caixa de seleção, caixa de entrada), nada é feito.
double getXcoord(String objName)	Retorna a coordenada cartesiana x do objeto com o nome fornecido. Nota: retorna 0 se o objeto não for um ponto ou um vetor.
double getYcoord(String objName)	Retorna a coordenada cartesiana y do objeto com o nome fornecido. Nota: retorna 0 se o objeto não for um ponto ou um vetor.

Fonte: Criada pelo autor com dados retirados de [22].

O *JavaScript* utiliza outros comandos de *Scripting*, apresentados na exemplificação do programa, em *JavaScript*, da tesselação supracitada e utilizados nas modelagens deste trabalho. Veja a Tabela 1.2.

Tabela 1.2: Outros comandos do GeoGebra que através do *JavaScript* são utilizados na modelagem.

Outros comandos do *JavaScript*	**Descrição**
Delete(<Object>)	Exclui o objeto e todos os seus objetos dependentes.
TurtleForward(<Turtle>, <Distance>)	A tartaruga avança uma determinada distância.
TurtleLeft(<Turtle>, <Angle>)	A tartaruga vira para a esquerda em um determinado ângulo.
TurtleRight(<Turtle>, <Angle>)	A tartaruga vira para a direita em um determinado ângulo.
SetCoords(<Object>, <x>, <y>)	Altera as coordenadas cartesianas de objetos livres. Este comando usa valores das coordenadas, não suas definições, portanto o objeto permanece livre.
StartAnimation( )	Retoma todas as animações se elas estiverem pausadas.

Fonte: Criada pelo autor com dados retirados de [22].

A tartaruga do GeoGebra e o *JavaScript* formam a parceria computacional usufruída para a modelagem matemática dos sistemas de Lindenmayer, os quais são explanados no Capítulo 2 a seguir.

Capítulo 2

Sistemas de Lindenmayer

"Uma tarefa central da biologia do desenvolvimento é descobrir o algoritmo subjacente ao longo do desenvolvimento."
Aristid Lindenmayer [30] (1974).

Originalmente proposto pelo biólogo Aristid Lindenmayer em 1968 como base para uma teoria axiomática do desenvolvimento biológico, os sistemas de Lindenmayer são sistemas dinâmicos discretos. Estes sistemas constituem a ferramenta teórica fundamental para a modelagem matemática computacional neste trabalho. Estes tipos de sistemas são denominados L-sistemas. cujo objetivo básico é definir objetos complexos substituindo partes simples do objeto utilizando um conjunto de regras de produção recursiva.

Os L-sistemas podem ser modelados matematicamente e computacionalmente de várias formas. A escolhida neste estudo é através do *turtle graphics* (veja Seção 1.2). É por isso que a caracterização teórica dos L-sistemas apresentada na Seção 2.1 está diretamente associada com a metodologia do *turtle graphics*.

2.1 Caracterização dos L-sistemas

A fim de caracterizar os L-sistemas utilizados, apresentam-se as seguintes definições.

Definição 2.1. Um conjunto finito V é um alfabeto de um L-sistema se contém pelo menos algum dos seguintes elementos:

- a letra F que, na linguagem do *turtle graphics*, significa andar para frente e desenhar um segmento de reta, cujo comprimento pode ser fixado na modelagem ou mudado por uma regra de produção.
- a letra f que significa andar para frente sem desenhar um segmento de reta, cujo comprimento pode ser fixado na modelagem ou mudado por uma regra de produção.
- o símbolo $+$ que significa girar, em sentido horário, um ângulo que pode ser fixado na modelagem ou mudado por uma regra de produção.
- o símbolo $-$ que significa girar, em sentido anti-horário, um ângulo que pode ser fixado na modelagem ou mudado por uma regra de produção.

- As letras maiúsculas $X, Y, Z, \ldots$ também conhecidas como variáveis do L-sistema, são componentes de V a serem substituídas, através das regras de produção do L-sistema, sem nenhum valor de movimento para a tartaruga. As variáveis não podem assumir a letra F, letra maiúscula correspondente a um movimento da tartaruga.

- O símbolo [que armazena uma posição específica em que a tartaruga se encontra no plano. Este símbolo é conhecido como símbolo de empilhamento. Para mais detalhes computacionais da dinâmica deste símbolo veja Capítulo 5.

- O símbolo] resgata a posição armazenada pelo símbolo [correspondente, ou seja, desempilha.

- O conjunto C que contém as constantes. Nas modelagens apresentadas, essas constantes podem ser: o comprimento do passo, a medida de um ângulo, a medida de contração ou expansão do passo.

Definição 2.2. O conjunto V^+ de um L-sistema é formado por todas as sequências finitas de elementos de V. Isto é, o conjunto de todas as funções $\mathbb{I} \subset \mathbb{N} \to V^+$, em que, $\mathbb{I} = 1, 2, \ldots, m$ é um conjunto ordenado.

Definição 2.3. O axioma de um L-sistema é um elemento de V^+, denotado por ω.

Definição 2.4. Uma regra de produção ou substituição de um L-sistema é um elemento do conjunto $V \times V^+$. Se $(v,\ v^+) \in V \times V^+$, esta regra é denotada como $v \to v^+$, sendo v o antecessor e v^+ o sucessor da regra de produção. O conjunto de todas as regras de produção é denotado por P.

Note que $P \subset V \times V^+$. Se nenhuma regra de substituição é especificada explicitamente para uma dada antecessora $v \in V$, a regra de substituição identidade $v \to v$ é considerada pertencente ao conjunto de regras de substituição P. Ou quando um símbolo não é substituído por nenhum outro, nem por ele mesmo, utiliza-se a notação $v \to \emptyset$ [44].

Definição 2.5. Um L-sistema é uma tripla ordenada $G = (V,\ \omega,\ P)$, em que, V é o alfabeto, ω é o axioma e P o conjunto de todas as regras de produção do L-sistema.

2.2 Ilustração de um L-sistema

Neste momento, explana-se melhor um L-sistema atráves da seguinte ilustração fictícia.
Suponha $G = (V, \omega, P)$, em que:

- Alfabeto: $V = \{F, +, -, \{\delta, 1/2\}\}$, em que, δ é um ângulo fixo medido em radianos ou graus e $1/2$ representa a contração do passo unitário inicial a cada iteração que, na dinâmica dos L-sistemas, é chamada de geração.

- Axioma: $\omega = F$.

- Regras de Produção: $P = \{F \to F - F + F - F+,\ + \to +,\ - \to -\}$.

Assim, as duas primeiras iterações são:

$$\underbrace{F}\quad ;$$
$$F-F+F-F+$$

$$\underbrace{F}\quad - \quad \underbrace{F}\quad + \quad \underbrace{F}\quad - \quad \underbrace{F}\quad +.$$
$$F-F+F-F+ \quad F-F+F-F+ \quad F-F+F-F+ \quad F-F+F-F+$$

Observe que o comprimento do passo atual é a metade do passo anterior.

A fim de ilustrar esse processo, veja Figura 2.1.

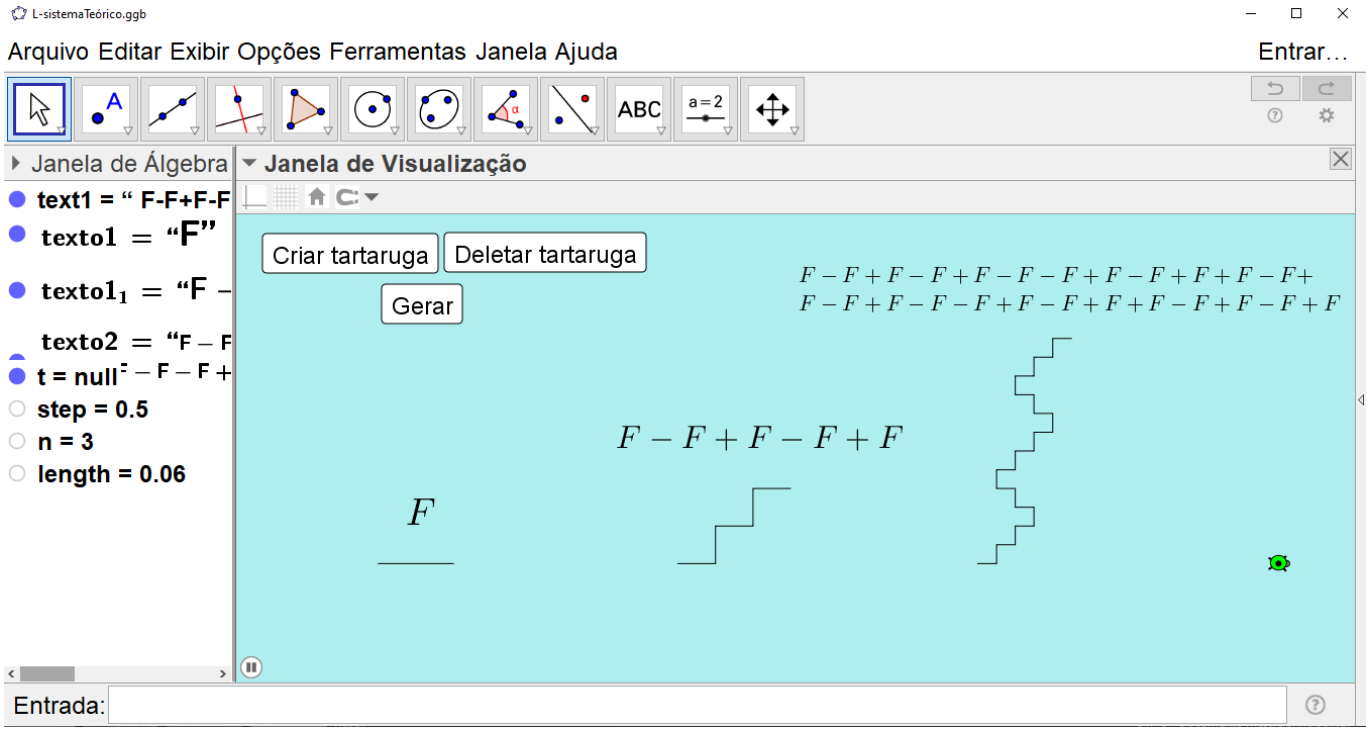


Figura 2.1: Interface do GeoGebra com a modelagem do L-sistema dado, com ângulo $\delta = \frac{\pi}{2}$, das 1ª, 2ª e 3ª gerações, com suas respectivas substituições.

Nos próximos capítulos são definidos L-sistemas dentro de várias áreas da matemática. O Capítulo 3 refere-se às modelagens de objetos fractais através dos L-sistemas.

Capítulo 3

Alguns Fractais Modelados via L-sistemas

"Por que a geometria é frequentemente descrita como 'fria' e 'seca'?
Uma razão reside na sua incapacidade de descrever a forma de uma nuvem, uma montanha, um litoral, ou uma árvore."
Benoît Mandelbrot [32] (1982).

Entre os L-sistemas implementados matematicamente e computacionalmente, encontram-se os que modelam os objetos matemáticos denominados fractais. Este termo, fractal, foi criado em 1975 por Benoît Mandelbrot, matemático francês nascido na Polônia a partir do adjetivo latino *fractus*, do verbo *frangere*, que significa quebrar, ou seja, criar fragmentos irregulares [32]. Mandelbrot que desenvolveu a geometria fractal na década de 1970, define em seu brilhante livro, intitulado *The Fractal Geometry of Nature*, o que ele chama de uma "nova geometria da natureza" que segundo ele mesmo diz:

> [...] descreve muitas das irregularidades e padrões fragmentados ao nosso redor, e leva a teorias completas, identificando uma família de formas que chamo de fractais. Os fractais mais úteis envolvem acaso e ambas as suas regularidades e suas irregularidades são estatísticas. Além disso, as formas descritas aqui tendem a ser escaláveis, implicando que o grau de sua irregularidade e/ou a fragmentação é idêntica em todas as escalas. [32]

Uma das principais características e propriedades de um fractal é a auto-semelhança [32] que, de forma sucinta e intuitiva, é quando, uma parte de escala reduzida de um objeto é ampliada e se assemelha com o todo. O motivo pelo qual esta metodologia matemática e computacional consegue representar e determinar estruturas de fractais é, como descrito no Capítulo 2, a capacidade dos L-sistemas de definir objetos complexos substituindo sucessivamente partes de um objeto simples, usando um conjunto de regras de produção recursiva [44].

A fim de modelar dois fractais famosos, utiliza-se a combinação da dinâmica dos L-sistemas com a tartaruga do GeoGebra e o *JavaScript*. Desta forma, é introduzido, na Seção 3.1 o conjunto de Cantor entendido como fractal e na Seção 3.2 o floco de neve de Koch, com suas respectivas estruturas dadas por um L-sistema, suas codificações e simulações.

3.1 O Conjunto de Cantor

O conjunto de Cantor foi descoberto em 1874 por Henry Smith [47] e introduzido pelo matemático alemão Georg Cantor em 1882 [8]. O conjunto de Cantor possui algumas propriedades distintas: é fechado, limitado, de interior vazio, não-enumerável, de conteúdo nulo, entre outras curiosidades matemáticas [29]. Além disso, é conhecida a propriedade de auto-similaridade do conjunto de Cantor que é explorada nessa seção.

Denotando por $K \subset [0,1]$, a construção do conjunto de Cantor é feita do seguinte modo:

- Retira-se do intervalo $[0,1]$ seu terço médio aberto $\left(\frac{1}{3}, \frac{2}{3}\right)$;
- Retira-se o terço médio aberto de cada um dos intervalos restantes $\left[0, \frac{1}{3}\right]$ e $\left[\frac{2}{3}, 1\right]$ são eliminados. Sobrando, então $\left[0, \frac{1}{9}\right] \cup \left[\frac{2}{9}, \frac{1}{3}\right] \cup \left[\frac{2}{3}, \frac{7}{9}\right] \cup \left[\frac{8}{9}, 1\right]$;
- Retira-se o terço médio aberto de cada um desses quatro intervalos e repete-se o processo.

O conjunto K dos pontos não retirados é o conjunto de Cantor [29].

A partir desta construção do conjunto de Cantor, fez-se a sua modelagem com o auxílio da tartaruga do GeoGebra com o *JavaScript*. Desse modo, a estrutura do L-sistema do conjunto de Cantor $G = (V, \omega, P)$ é dada por:

Alfabeto: $V = \{F, f, \{1/3\}\}$;

Axioma: $\omega = F$;

Regras de Produção: $P = \{F \to FfF, f \to fff\}$.

Salienta-se, como explicado no Capítulo 2, o passo F contrai $1/3$ de seu comprimento em cada iteração. Uma simulação, das 1^a à 6^a gerações, pode ser vista na Figura 3.1 seguidamente de sua codificação.

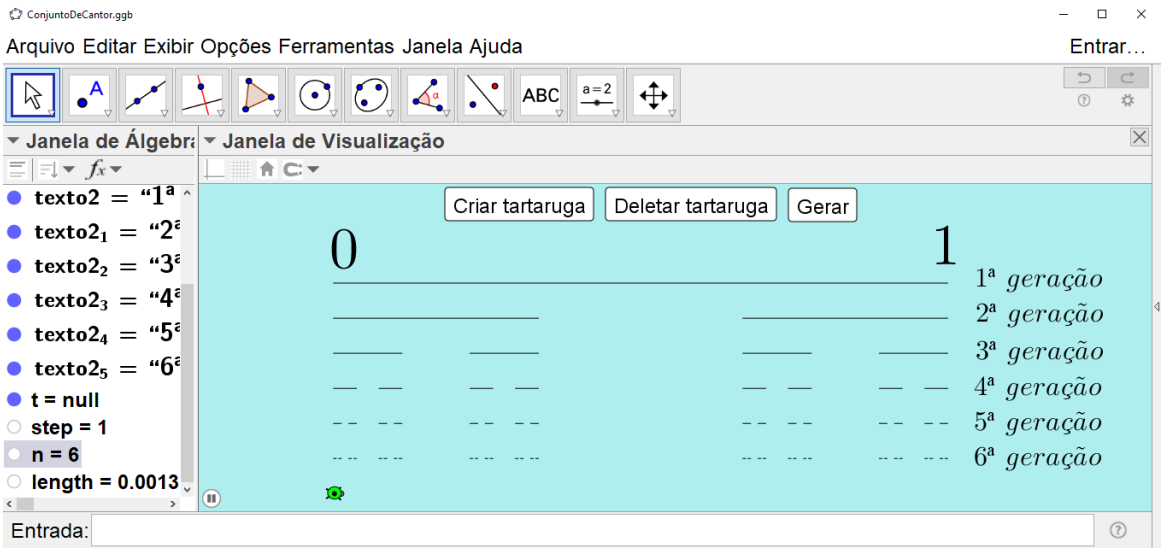


Figura 3.1: Modelagem do conjunto de Cantor da 1^a geração até a 6^a geração, utilizando a tartaruga do GeoGebra com o *JavaScript*.

Note que, na Figura 3.1 são mostrados os segmentos cujos extremos são os pontos do conjunto de Cantor. O *turtle graphics* do GeoGebra possui a limitação de não desenhar apenas um ponto, desta forma a modelagem é feita através dos intervalos que definem o conjunto de Cantor.

```
function ggbOnInit() {}
var sentencaMostrada;
var sentenca;
var proximaSentenca;

function posicionar() {
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[]");
    sentencaMostrada = "F";
    sentenca = "F";
    proximaSentenca = " ";
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[t]");
}

function Fcom() {
    ggbApplet.evalCommand("length=step/3^n");
    ggbApplet.evalCommand("TurtleForward[t, length]");
}

function fcom() {
    ggbApplet.evalCommand("TurtleUp[t]");
    ggbApplet.evalCommand("TurtleForward[t, length]");
    ggbApplet.evalCommand("TurtleDown[t]");
}

function turtle() {
    ggbApplet.setTextValue("text", sentencaMostrada);
    proximaSentenca = " ";
    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
```

```
        if (atual == "F") {
            Fcom();
        } else if (atual == "f") {
            fcom();
        }
    }
    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            proximaSentenca = proximaSentenca + "FfF";
        } else if (atual == "f") {
            proximaSentenca = proximaSentenca + "fff";
        }
    }
    sentencaMostrada = proximaSentenca;
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
    ggbApplet.evalCommand("SetCoords[t,0,-n]");
}
```

Dando uma breve descrição das linhas de código acima, pode-se notar, de início, a criação de três variáveis: $sentencaMostrada$, $sentenca$ e $proximaSentenca$. A primeira variável tem a função de mostrar na "Janela de Álgebra", a cada iteração, o produto de todas as regras de produção já realizadas. A segunda representa o axioma, ou seja, a sentença inicial que é substituída por outra a cada iteração. A terceira representa a proxima sentença que é substituída no axioma na iteração seguinte.

Foram definidas cinco funções para organização do programa. A primeira, $posicionar()$, é a função que é realizada ao clicar no botão "Criar tartaruga" e inicia a modelagem criando a tartaruga e a prepara para realizar o axioma que, neste caso, é F.

A segunda, $deletarTartaruga$, é a função que deleta toda a modelagem e é realizada quando se clica no botão "Deletar tartaruga".

A terceira, $Fcom()$ é a função que é utlizada dentro da função $turtle()$ e determina a movimentação para frente da tartaruga num passo de comprimento $step/3^n$. Este $step$ é definido dentro do botão "Criar tartaruga" e tem o valor de 1, e o n é a potência do 3 que se altera a cada geração da modelagem, ou seja, o passo da tartaruga contrai a cada iteração a uma razão de $\frac{1}{3}$.

A quarta função, $fcom()$, age na tartaruga com o propósito de fazê-la andar sem desenhar.

A quinta função, $turtle()$, é realizada ao clicar no botão "Gerar" e determina toda a movimentação da tartaruga. Com o auxílio de dois comandos $for()$, a tartaruga se movimenta quando o programa realiza o

primeiro $for()$, ou seja, este primeiro comando interpreta símbolo por símbolo da primeira *string* $sentenca$ atual, que na primeira iteração simboliza o axioma e, se o símbolo interpretado for um F, a tartaruga anda desenhando, caso contrário, se for um f, a tartaruga anda sem desenhar. O segundo $for()$ faz as devidas substituições para as próximas gerações. Neste caso, cada F é substituído por FfF e cada f é substituído por fff, ou seja, $F \to FfF$ e $f \to fff$. Logo após, substitui a $sentenca$ atual por $proximaSentenca$ que a tartaruga realizará na próxima geração. Incrementa a variável n em uma unidade e altera as coordenadas da tartaruga para $[0, -n]$. Além disso, a cada clique no botão "Gerar" representa uma geração, ou seja, o primeiro clique modela a primeira geração (axioma), o segundo clique modela a segunda geração e assim sucessivamente.

3.2 O Floco de Neve de Koch

O famoso floco de neve de Koch [27] é um caso particular da curva de Koch. O conceito de curva de Koch aparece pela primeira vez num artigo de 1906, intitulado *Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes* [27], de autoria do matemático sueco Helge von Koch. Vale ressaltar que esta curva foi um dos primeiros fractais a serem descritos.

Segundo Prusinkiewicz e Sandness [43], em seu artigo publicado sob o de título *Koch curves as attractors and repellers*:

> [...] a construção de Koch consiste em substituir recursivamente as arestas de um polígono arbitrário (denominado iniciador) por um polígono aberto (o gerador), reduzido e deslocado de modo a ter os mesmos pontos finais que os do intervalo a ser substituído. (A construção original foi limitada à definição da agora famosa curva "floco de neve").

A partir da descrição supracitada, há várias maneiras de modelar o fractal de Koch. Dispondo a modelagem do floco de neve de Koch na estrutura dos L-sistemas, ou seja, considerando uma terna $G = (V, \omega, P)$, tem-se que:

Alfabeto: $V = \{F, +, -, \{\delta_1 = 2\pi/3, \delta_2 = \pi/3, 1/3\}\}$;

Axioma: $\omega = -F + F + F + +-$;

Regras de Produção: $P = \{F \to F - F + F - F, + \to +, - \to -\}$.

Note que o passo F é contraído pela constante $1/3$ a cada iteração.

Seu código de programação em *JavaScript*, e suas simulações serão exibidas a seguir, com uma explicação detalhada de sua codificação.

```
function ggbOnInit() {}
var proximaSentenca
var sentenca;
```

```
function posicionar() {
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.setTextValue("text1", " ");
    sentenca = "-F+F+F++-";
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[t]");
}

function Fcom() {
    ggbApplet.evalCommand("length=1/(3^n)")
    ggbApplet.evalCommand("TurtleForward[t,length]");
}

function Dcom() {
    ggbApplet.evalCommand("TurtleRight[t, 2*pi/3]");
}

function Ecom() {
    ggbApplet.evalCommand("TurtleLeft[t, pi/3]");
}

function moverTartaruga() {
    proximaSentenca = " ";
    ggbApplet.setTextValue("text1", sentenca);
    ggbApplet.evalCommand("Delete[t]");
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[]");
    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            Fcom();
        } else if (atual == "+") {
            Dcom();
        } else if (atual == "-") {
```

```
            Ecom();
        }
    }
    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            proximaSentenca = proximaSentenca + "F-F+F-F";
        } else if (atual == "+") {
            proximaSentenca = proximaSentenca + "+";
        } else if (atual == "-") {
            proximaSentenca = proximaSentenca + "-";
        }
    }
    ggbApplet.evalCommand("n=n+1");
    sentenca = proximaSentenca;
}
```

Analisando os comando acima, do início, percebe-se que são criadas duas variáveis de nomes $sentenca$ e $proximaSentenca$, as quais representam, respectivamente, o axioma (na primeira iteração) e as regras de produção que serão substituidas na variável $sentenca$.

Nesta modelagem, a tartaruga precisa fazer giros no sentido horário e anti-horário (diferente da programação do conjunto de Cantor, o qual ela somente andava em linha reta). Para isso, é necessária a criação de duas funções que determinam para qual sentido é seu giro e qual o valor do ângulo que a tartaruga deve girar.

Da mesma forma que no código do conjunto de Cantor, neste código do floco de neve de Koch são definidas seis funções:

1. $posicionar()$: Tem a função de iniciar a tartaruga e, além disso, prepará-la para realizar o axioma dado por: $-F+F+F++-$. Esta função é realizada ao clicar no botão "Criar tartaruga".

2. $deletarTartaruga()$: é a função que deleta toda a modelagem e é realizada quando se clica no botão "Deletar tartaruga".

3. $Fcom()$: função que é utlizada dentro da função $moverTartaruga()$ e determina a movimentação para frente da tartaruga num passo de tamanho $\frac{1}{3^n}$. A variável n é a potência do 3 que se altera a cada geração da modelagem, ou seja, o passo da tartaruga contrai a cada iteração a uma razão de $\frac{1}{3}$.

4. $Dcom()$: função que também é utilizada dentro da função $moverTartaruga()$. Determina que a tartaruga deve realizar um giro no sentido horário, com um ângulo de $\frac{2\pi}{3}$ radianos.

5. $Ecom()$: é a função que também é utilizada dentro da função $moverTartaruga()$. Determina que a tartaruga deve realizar um giro no sentido anti-horário, com um ângulo de $\frac{\pi}{3}$ radianos.

6. $moverTartaruga()$: é realizada ao clicar no botão "Gerar" e determina toda a movimentação da tartaruga. Logo no início desta função, foram definidos comandos como:

$$ggbApplet.evalCommand("Delete[t]"),$$
$$ggbApplet.evalCommand("t = Turtle[]"),$$
$$ggbApplet.evalCommand("StartAnimation[]"),$$

nesta mesma ordem e que possuem o papel de, no início de cada iteração, excluir toda a modelagem da geração anterior e criar outra tartaruga, de mesmo nome que a anterior, ou seja, de nome t, para inicar a próxima geração. Estes comandos foram definidos para que a "Janela de Visualização" do GeoGebra não fique poluída.

Com o auxílio de dois comandos $for()$, a tartaruga se movimenta e gira seu corpo nos sentidos horário e anti-horário, quando o programa realiza o primeiro $for()$, ou seja, este primeiro comando interpreta símbolo por símbolo da primeira *string* $sentenca$ atual, que na primeira iteração simboliza o axioma. Se o símbolo interpretado for um F, a tartaruga anda para frente desenhando, caso o símbolo for um $+$, retorna a função $Dcom()$ fazendo com que a tartaruga gire seu corpo no sentido horário e, se o símbolo for um $-$ retorna a função $Ecom()$ fazendo com que a tartaruga gire seu corpo no sentido anti-horário.

O segundo $for()$ realiza as devidas substituições para as próximas iterações. Neste caso, cada F é substituído por $F - F + F - F$, e quando for $+$ ou $-$, eles permanecem iguais, segundo as regras de produção do L-sistema do floco de neve de Koch. Logo após, substitui a $sentenca$ atual para $proximaSentenca$ que a tartaruga realizará na próxima geração. Incrementa a variável n em uma unidade e representa a geração a qual a modelagem se encontra. Além disso, a codificação não altera as coordenadas da tartaruga pois, como a cada nova geração, a anterior é deletada, não há necessidade de redefinir suas coordenadas.

Lembre-se que, a cada clique no botão "Gerar" representa uma geração, ou seja, o primeiro clique modela a primeira geração (axioma), o segundo clique modela a segunda geração e assim sucessivamente.

Veja Figuras 3.2, 3.3, 3.4 para melhor visualização de todo o processo da programação.

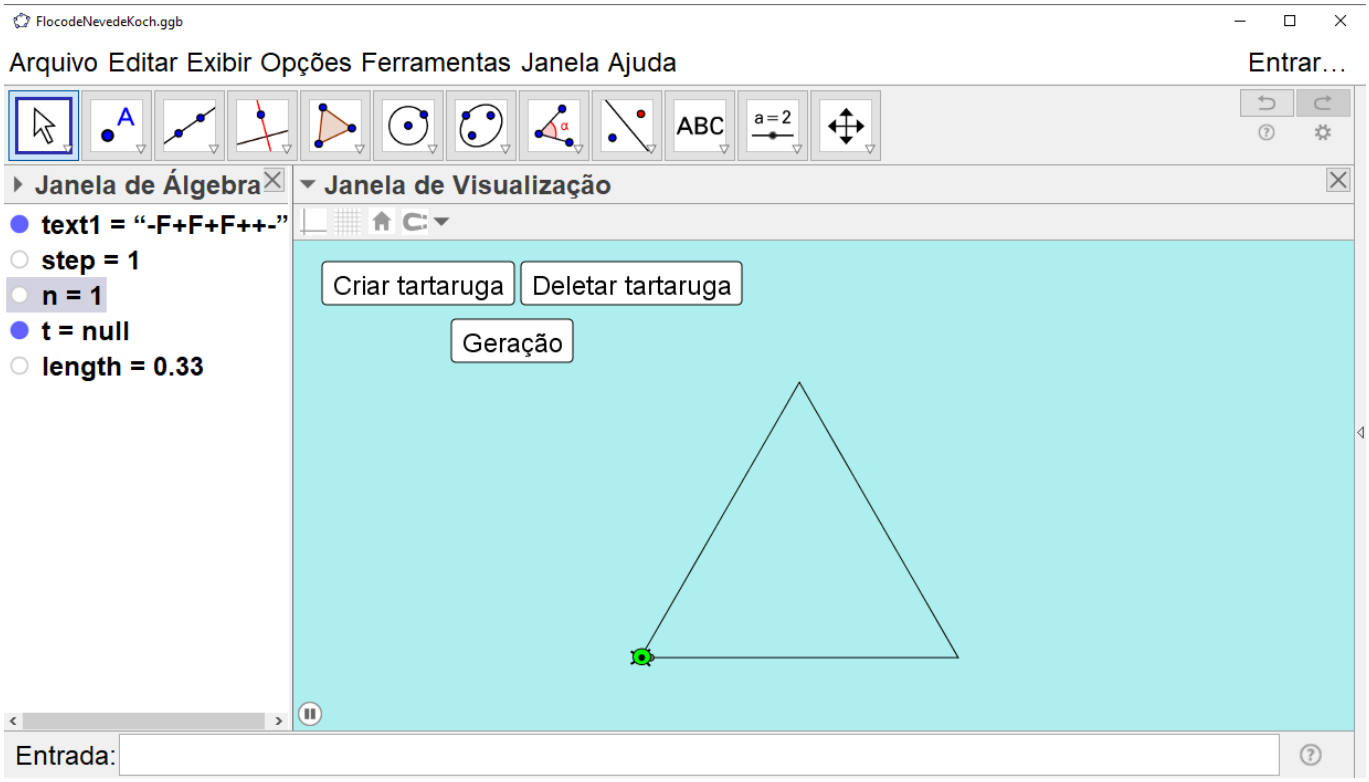


Figura 3.2: Interface do GeoGebra com a modelagem da primeira geração ($n = 1$ na "Janela e Álgebra") do floco de neve de Koch.

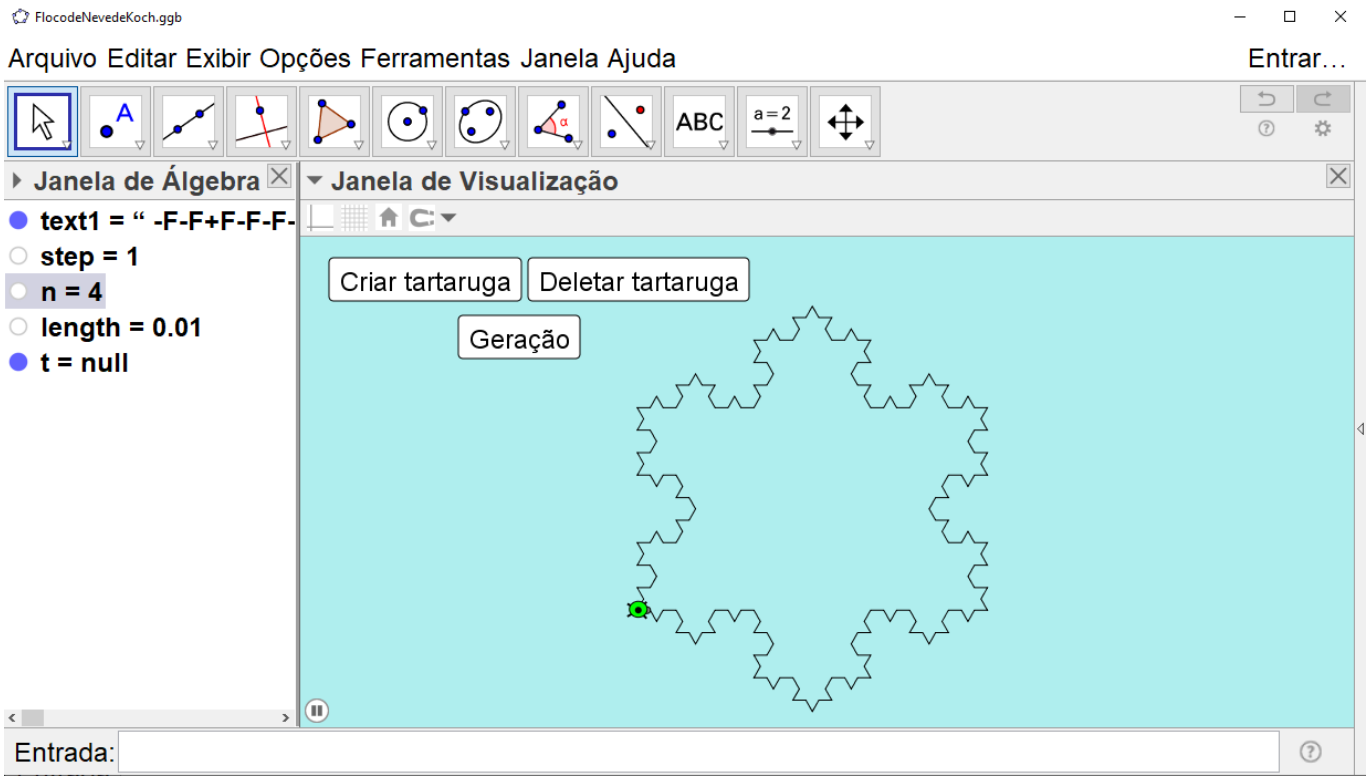


Figura 3.3: Interface do GeoGebra com a modelagem da quarta geração ($n = 4$ na "Janela e Álgebra") do floco de neve de Koch.

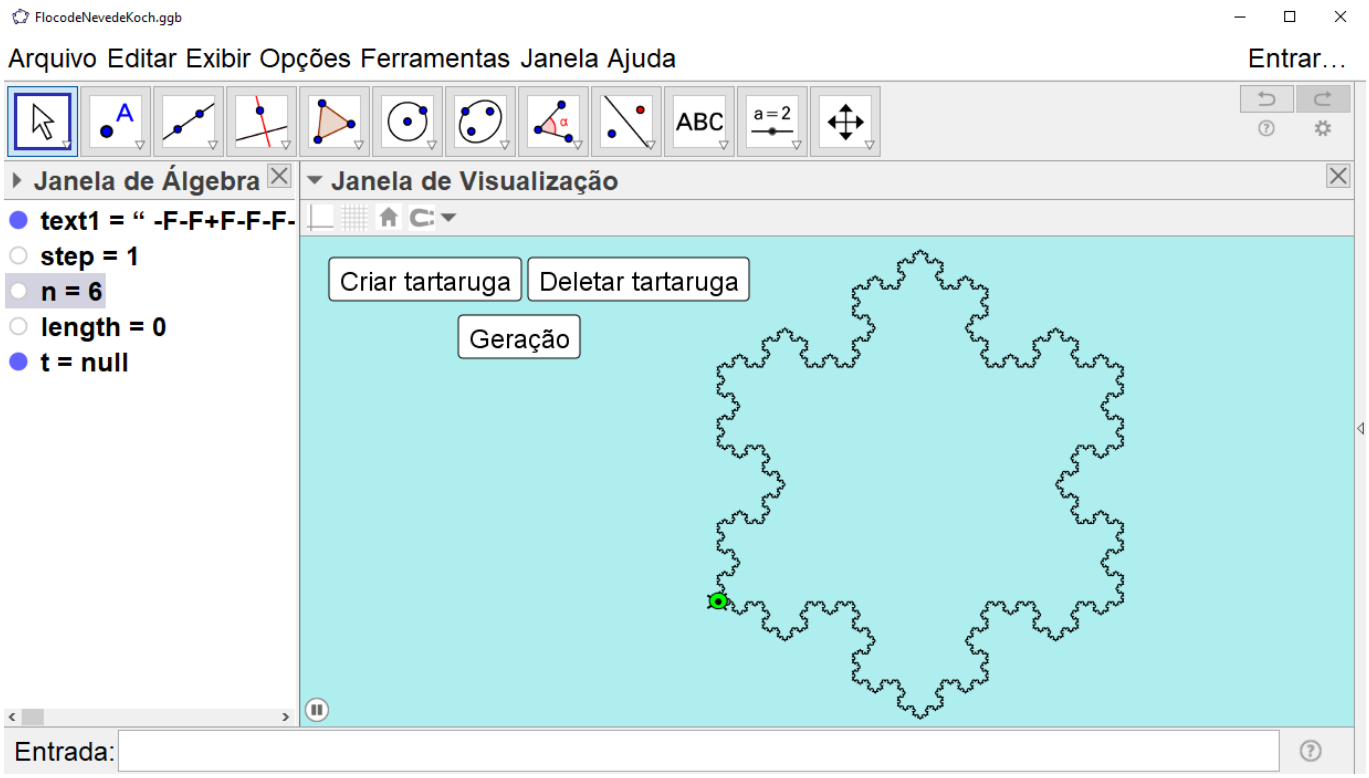


Figura 3.4: Interface do GeoGebra com a modelagem da sexta geração ($n = 6$ na "Janela e Álgebra") do floco de neve de Koch.

Note que na "Janela de Álgebra" das Figuras 3.2, 3.3, 3.4 possui um $text1$ o qual exibe a substituição feita da geração anterior para a atual, e a variável n que é incrementada no final do programa, também serve para controlar as gerações. No caso da Figura 3.2, como $n = 1$, então encontra-se na primeira iteração, na Figura 3.3, como $n = 4$, então encontra-se na quarta geração e assim sucessivamente.

A metodologia utilizada nesta etapa da investigação mostra a eficiência da comunicação entre a linguagem *JavaScript* e a tartaruga em termos de movimentos. Na próxima etapa o desafio é de incorporar variáveis ao alfabeto. Os resultados obtidos são apresentados na modelagem das curvas de preenchimento espacial no Capítulo 4.

Capítulo 4

L-sistemas e as Curvas FASS

"A ambiguidade da linguagem é a principal fonte de problemas da filosofia. Por isso é da maior importância examinar atentamente as próprias palavras que usamos."

Giuseppe Peano [37] (1889).

4.1 As Curvas FASS

Uma curva de preenchimento espacial $n-$dimensional é uma função contínua sobrejetiva de $[0, 1]$ para $[0, 1]^n$, o hipercubo de dimensão n. Giuseppe Peano em 1890 descreveu tais mapeamentos para $n = 2$ e $n = 3$ [38] dando origem à, denominada, curva de Peano. Versões dessa curva surgiram a seguir, entre elas, a de David Hilbert em 1891 [20] e a criada por Eliakim Moore em 1900 [35], uma variante da curva de Hilbert [45], [12]. Além disso, outras curvas de preenchimento espacial, também denominadas, em geral, de curvas de Peano [18], foram encontradas e aplicadas em diversas áreas de conhecimento: gerenciamento de nuvem, mineração de dados de alta performance, sistemas de informações geográficas, processamento de imagens, bancos de dados, projeto de circuitos, criptologia, algoritmo evolucionário, balanceamento de carga em computação paralela, melhoramento de cache, utilização em cálculos em grandes matrizes, métodos de elementos finitos, compressão de imagens, otimização combinatória, entre outras [1], [2], [3], [4], [5], [6], [7], [17].

Em particular, dentre as curvas de preenchimento espacial possuem a classe denominada **FASS**, um acrônimo para o conceito space-**F**illing, self-**A**voiding, **S**imple, and self-**S**imilar em inglês, significando que as curvas preenchem o espaço, são auto-similares, contínuas e sem auto-interseção [18]. Ou seja, são curvas que gozam da propriedade que, se no plano $\mathbb{R}^2$ estiver definido uma malha quadriculada unitária, as curvas FASS percorrem exatamente uma vez cada quadrado da malha. Existem diferentes métodos de gerar essas curvas: métodos fractais, por arimetização [45], L-sistemas [44], entre outros puramente computacionais. A auto-similaridade provém de definir o domínio dessas curvas em $[0, 1] \times [0, 1]$ e efetuar o preenchimento desse quadrado contraindo a cada iteração o comprimento dos segmentos que compõe a curva [35].

4.2 Peano, Hilbert e Moore: Três Curvas FASS Entrelaçam Três Mentes Brilhantes

Como mencionado anteriormente, Peano, Hilbert e Moore, nesta ordem cronológica entrelaçam-se historicamente através das curvas descritas por essas três mentes brilhantes.

Nesta seção, efetua-se a modelagem matemática e computacional destas três curvas através dos L-sistemas e do *turtle graphics* do GeoGebra. De fato, na Tabela 4.1 se encontram todas as estruturas dos L-sistemas das curvas de Peano, Hilbert e Moore, ou seja as três ternas que compõem, nesta ordem, o alfabeto V, o axioma ω e o conjunto das regras de produção P.

Tabela 4.1: As ternas G_P, G_H e G_M correspondentes à curva de Peano, Hilbert e Moore, respectivamente.

	V	ω	P
G_P	$\{F,+,-,\{\delta=\pi/2\}\}$	F	$+\to+, -\to-$; $F\to F+F-F-F-F+F+F+F-F.$
G_H	$\{F,+,-,X,Y,\{\delta=\pi/2\}\}$	X	$+\to+, -\to-, F\to F$; $X\to -YF+XFX+FY-$; $Y\to +XF-YFY-FX+.$
G_M	$\{F,+,-,X,Y,\{\delta=\pi/2\}\}$	$XFX+F+XFX$	$+\to+, -\to-, F\to F$; $X\to -YF+XFX+FY-$; $Y\to +XF-YFY-FX+.$

A partir da Tabela 4.1, mostra-se a codificação de cada curva, iniciando pela curva de Peano, representada pelo L-sistema $G_P=(V,\omega,P)$.

Sua codificação é dada a seguir:

```
function ggbOnInit() {}

var sentenca;
var proximaSentenca;

function posicionar() {
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    sentenca = "F";
    proximaSentenca = " ";
}
```

```
function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[t]");
}

function Fcom() {
    ggbApplet.evalCommand("length=step");
    ggbApplet.evalCommand("TurtleForward[t, length]");
}

function Dcom() {
    ggbApplet.evalCommand("TurtleRight[t, pi/2]");
}

function Ecom() {
    ggbApplet.evalCommand("TurtleLeft[t, pi/2]");
}

function turtle() {
    ggbApplet.setTextValue("text", sentenca);
    ggbApplet.evalCommand("Delete[t]");
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    proximaSentenca = " ";
    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            Fcom();
        } else if (atual == "+") {
            Dcom();
        } else if (atual == "-") {
            Ecom();
        }
    }

    for (var i = 0; i < (sentenca.length); i++) {
```

```
        atual = sentenca.charAt(i);
        if (atual == "F") {
            proximaSentenca = proximaSentenca + "F+F-F-F-F+F+F+F-F";
        } else if (atual == "+") {
            proximaSentenca = proximaSentenca + "+";
        } else if (atual == "-") {
            proximaSentenca = proximaSentenca + "-";
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
}
```

A interpretação deste código é simples. É dividido em seis funções, sendo três delas: $Fcom()$, $Dcom()$ e $Ecom()$ funções utilizadas dentro da função $turtle()$, as quais, a primeira realiza a movimentação da tartaruga para frente com passo de tamanho $lenght$, definido na aba "Ao Clicar" do botão "Criar tartaruga", a segunda realiza um giro no sentido horário com um ângulo de $\frac{\pi}{2}$ radianos. E a terceira realiza um giro no sentido anti-horário com um ângulo também de $\frac{\pi}{2}$ radianos.

As outras três funções que restaram: $posicionar()$, $deletarTartaruga()$ e $turtle()$ desempenham, basicamente, os mesmos papéis que as funções, com esses mesmos nomes, nas modelagens anteriores do Capítulo 3. A função $posicionar()$ inicia a tartaruga e, além disso, a prepara para realizar o axioma dado por: F. Esta função é realizada ao clicar no botão "Criar tartaruga". A função $deletarTartaruga()$ deleta toda a modelagem e é efetuada quando se clica no botão "Deletar tartaruga". E a função $turtle()$, além de excluir toda a modelagem e criar outra tartaruga de mesmo nome, quando se altera de geração, utiliza dois comandos $for()$. O primeiro $for()$ avalia a variável $sentenca$, definida no início da codificação, a qual armazena, na primeira iteração, o axioma F, assim, avaliando símbolo por símbolo, a tartaruga realiza determinada ação. O segundo $for()$ desempenha o papel de realizar todas as substituições dos símbolos que estão contidos na variável $sentenca$, de modo a seguir as regras de produção do L-sistema, armazenando essas substituições em $proximaSentenca$.

E, por fim, a variável $sentenca$ passa a armezenar o conteúdo da variável $proximaSentenca$ e o n é incrementado em uma unidade, o qual, novamente, é utilizado para exibir qual a geração a tartaruga está modelando.

O resultado desta modelagem matemática e computacional pode ser visualizado na Figura 4.1.

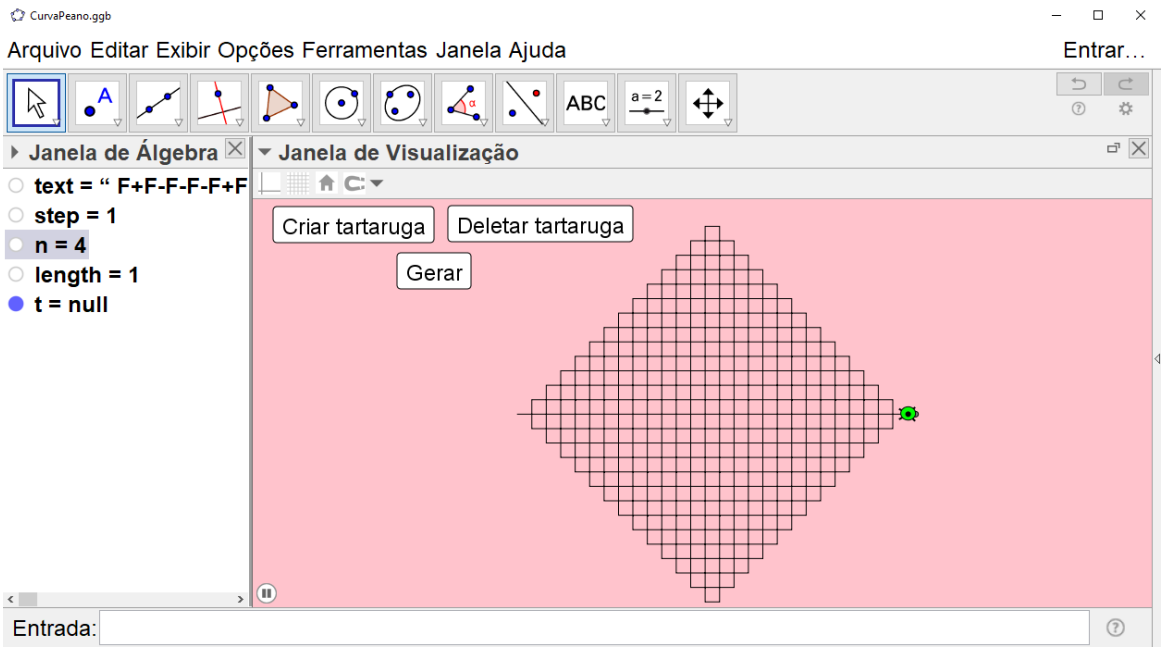


Figura 4.1: Modelagem Matemática e Computacional da curva de Peano com a interface do GeoGebra na 4^{a} geração ($n = 4$ na "Janela de Álgebra").

A estrutura do L-sistema $G_H = (V, \omega, P)$ da curva de Hilbert encontra-se na Tabela 4.1.

A partir disso, sua codificação é dada por:

```
function ggbOnInit() {}
var sentenca;
var proximaSentenca;
var X;
var Y;

function posicionar() {
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    sentenca = "X";
    proximaSentenca = " ";
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[t]");
}

function Fcom() {
    ggbApplet.evalCommand("length=step");
```

```
    ggbApplet.evalCommand("TurtleForward[t, length]");
}

function Dcom() {
    ggbApplet.evalCommand("TurtleRight[t, pi/2]");
}

function Ecom() {
    ggbApplet.evalCommand("TurtleLeft[t, pi/2]");
}

function turtle() {
    ggbApplet.setTextValue("text", sentenca);
    ggbApplet.evalCommand("Delete[t]");
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    proximaSentenca = " ";
    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            Fcom();
        } else if (atual == "+") {
            Dcom();
        } else if (atual == "-") {
            Ecom();
        }
    }

    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            proximaSentenca = proximaSentenca + "F";
        } else if (atual == "+") {
            proximaSentenca = proximaSentenca + "+";
        } else if (atual == "-") {
            proximaSentenca = proximaSentenca + "-";
```

```
        } else if (atual == "X") {
            proximaSentenca = proximaSentenca + "-YF+XFX+FY-";
        } else if (atual == "Y") {
            proximaSentenca = proximaSentenca + "+XF-YFY-FX+";;
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
}
```

A interpretação de sua codificação é análoga à da curva de Peano, com exceções de que, na curva de Hilbert, há a necessidade da criação de duas variáveis X e Y que, como foi visto no Capítulo 2 não possuem valor de movimento para a tartaruga, apenas para o controle das iterações e, além disso, seu axioma é dado por $sentenca = X$. A simulação desta curva pode ser visualizada na Figura 4.2.

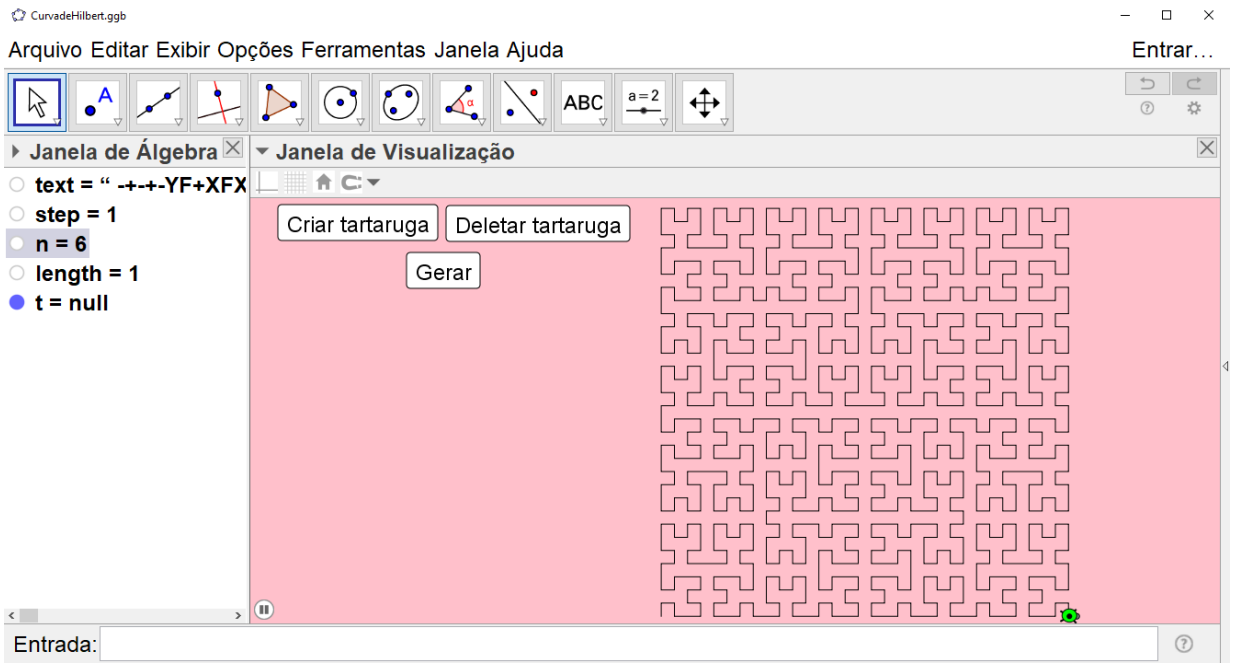


Figura 4.2: Modelagem Matemática e Computacional da curva de Hilbert com a interface do GeoGebra na 6ª geração ($n = 6$ na "Janela de Álgebra").

Por fim, a estrutura do L-sistema $G_M = (V, \omega, P)$ da curva de Moore encontra-se na Tabela 4.1.

Sua codificação é dada por:

```
function ggbOnInit() {}
var sentenca;
var proximaSentenca;
var X;
var Y;
```

```
function posicionar() {
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    sentenca = "XFX+F+XFX";
    proximaSentenca = " ";
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[t]");
}

function Fcom() {
    ggbApplet.evalCommand("length=step");
    ggbApplet.evalCommand("TurtleForward[t, length]");
}

function Dcom() {
    ggbApplet.evalCommand("TurtleRight[t, pi/2]");
}

function Ecom() {
    ggbApplet.evalCommand("TurtleLeft[t, pi/2]");
}

function turtle() {
    ggbApplet.setTextValue("text", sentenca);
    ggbApplet.evalCommand("Delete[t]");
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("TurtleLeft[t, pi/2]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    proximaSentenca = " ";
    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            Fcom();
```

```
        } else if (atual == "+") {
            Dcom();
        } else if (atual == "-") {
            Ecom();
        }
    }

    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            proximaSentenca = proximaSentenca + "F";
        } else if (atual == "+") {
            proximaSentenca = proximaSentenca + "+";
        } else if (atual == "-") {
            proximaSentenca = proximaSentenca + "-";
        } else if (atual == "X") {
            proximaSentenca = proximaSentenca + "-YF+XFX+FY-";
        } else if (atual == "Y") {
            proximaSentenca = proximaSentenca + "+XF-YFY-FX+";
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
}
```

Neste caso, a interpretação da codificação da curva de Moore é bem parecida com a da curva de Hilbert, até mesmo suas regras de produção são as mesmas, com as únicas exceções de que o axioma da curva de Moore é dado por $sentenca = XFX + F + XFX$ e, no início de cada nova geração, a tartaruga realiza um giro de $\frac{\pi}{2}$ radianos no sentido anti-horário. Este processo se dá pelo comando $ggbApplet.evalCommand(\text{“}TurtleLeft[t, pi/2]\text{”})$ definido no interior e no início da função $turtle()$.

Para sua visualização, veja Figura 4.3.

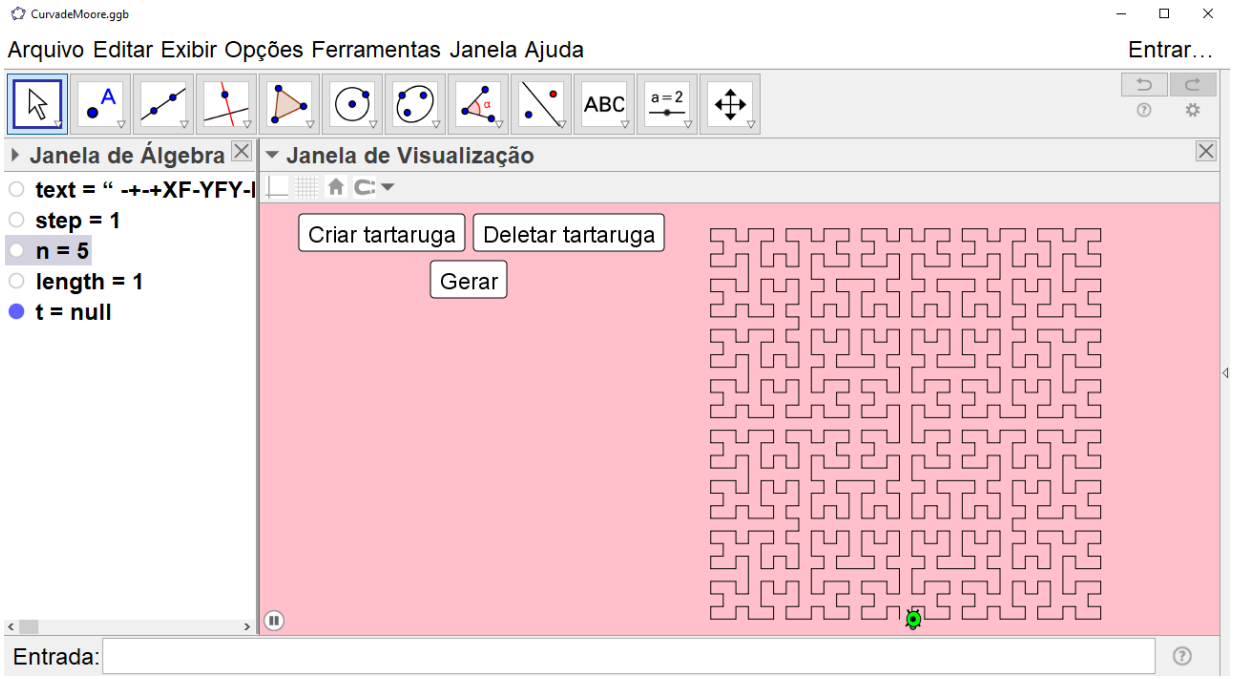


Figura 4.3: Modelagem Matemática e Computacional da curva de Moore com a interface do GeoGebra na 5^{a} geração ($n = 5$ na "Janela de Álgebra").

Percebe-se que as curvas de Hilbert e Moore são parecidas. Isto ocorre pois, a curva de Moore é uma variação da curva de Hilbert [45], de modo que se inicie e termine sua modelagem em pontos adjacentes [12].

4.3 A Curva do Dragão

A curva do Dragão, também conhecida como Dragão de *Harter–Heighway* ou Dragão *Jurassic Park*, foi investigada primeiro na NASA pelos físicos John Heighway, Bruce Banks, e William Harter. É uma curva bidimensional, o qual seu nome é dado "[...] provavelmente devido à semelhança de sua fronteira exterior com os tradicionais 'dragões' chineses", como é apontado por Crilly et al. [11].

Em seu trabalho, Tabachnikov [48], destaca a propriedade mais impressionante acerca desta curva:

Teorema 4.1. *A curva do Dragão não possui auto-interseções: isto é, se no plano $\mathbb{R}^2$ estiver definido uma malha quadriculada unitária, a curva do Dragão percorre exatamente uma vez cada quadrado da malha.*

Outra propriedade tão importante quanto a anterior, segundo Chang e Zhang [9], é que a fronteira topológica da curva do Dragão é um fractal por possuir a propriedade de auto-similaridade. Além disso, Chang e Zhang [9] afirmam que é uma curva de preenchimento espacial e é, nesse sentido, que a curva do Dragão pode ser representada como um L-sistema.

A curva do Dragão é modelada pelo L-sistema $G = (V, \omega, P)$, cujas componentes são:

Alfabeto: $V = \{F, +, -, X, Y, \{\delta = \pi/2\}\}$.

Axioma: $\omega = FX$.

Regras de Produção: $P = \{X \to X + YF+, Y \to -FX - Y, + \to +, - \to -, F \to F\}$.

Da mesma forma que foi feito nas modelagens computacionais anteriores, apresenta-se primeiro as linhas de comando do programa que modela a curva do Dragão, seguidas das simulações.

```
function ggbOnInit() {}
var sentenca;
var proximaSentenca;
var X;
var Y;

function posicionar() {
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    sentenca = "FX";
    proximaSentenca = " "
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[t]");
}

function Fcom() {
    ggbApplet.evalCommand("length=step");
    ggbApplet.evalCommand("TurtleForward[t, length]");
}

function Dcom() {
    ggbApplet.evalCommand("TurtleRight[t, pi/2]");
}

function Ecom() {
    ggbApplet.evalCommand("TurtleLeft[t, pi/2]");
}

function turtle() {
    ggbApplet.setTextValue("text", sentenca);
    ggbApplet.evalCommand("Delete[t]");
```

```
    ggbApplet.evalCommand("t=Turtle[]");
    ggbApplet.evalCommand("TurtleLeft[t, pi/2]");
    ggbApplet.evalCommand("StartAnimation[]");
    proximaSentenca = " ";
    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            Fcom();
        } else if (atual == "+") {
            Dcom();
        } else if (atual == "-") {
            Ecom();
        }
    }

    for (var i = 0; i < (sentenca.length); i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            proximaSentenca = proximaSentenca + "F";
        } else if (atual == "+") {
            proximaSentenca = proximaSentenca + "+";
        } else if (atual == "-") {
            proximaSentenca = proximaSentenca + "-";
        } else if (atual == "X") {
            proximaSentenca = proximaSentenca + "X+YF+";
        } else if (atual == "Y") {
            proximaSentenca = proximaSentenca + "-FX-Y";
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
}
```

A interpretação das linhas de código deste programa é análoga à interpretação feita nas modelagens anteriores.

De início, são criadas quatro variáveis, as duas primeiras: $sentenca$ e $proximaSentenca$ e duas últimas: X e Y varíaveis que auxiliam na evolução das gerações e não atuam e nem prejudicam nos movimentos

da tartaruga.

A primeira função, $posicionar()$, inicia a tartaruga e, além disso, a prepara para realizar o axioma dado por: FX.

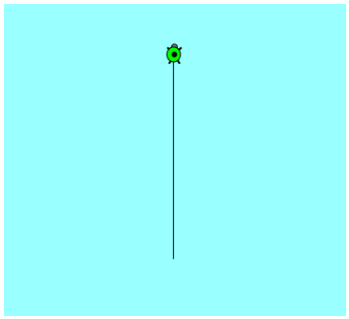
A função $turtle()$ é a função que realiza toda a movimentação da tartaruga. Note que, na 4^a linha de comando, possui o código $ggbApple.evalCommand(\text{“}TurtleLeft[t, pi/2]\text{”})$, cujo papel é realizar um giro inicial, na tartaruga, de $\frac{\pi}{2}$ radianos no sentido anti-horário a todo momento que inicia uma nova geração. Ainda no interior desta função, possui dois comandos $for()$, cujo segundo realiza as seguintes substituições:

$$\begin{aligned} F &\to F; \\ + &\to +; \\ - &\to -; \\ X &\to X + YF+; \\ Y &\to -FX - Y, \end{aligned}$$

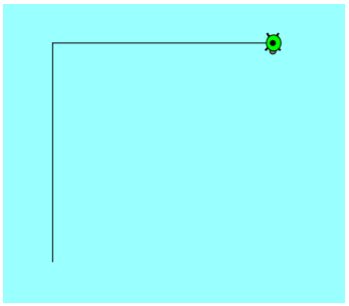
a cada nova geração. Cada substituição dessa é armazenada em $proximaSentenca$ e, o produto delas é armazenado em $sentenca$ para a tartaruga efetuar uma nova geração.

Assim, na primeira iteração, realiza-se o axioma FX, ou seja, a tartaruga anda uma unidade para frente. Na segunda iteração, faz-se a substituição $FX \to FX + YF+$, ou seja, ela anda uma unidade à frente, aguarda, gira no sentido horário, seguidamente de uma pausa para depois ir para frente uma unidade e terminar com um giro no sentido horário. Analogamente, se obtem a terceira iteração que é dada por $FX + YF + + - FX - YF+$.

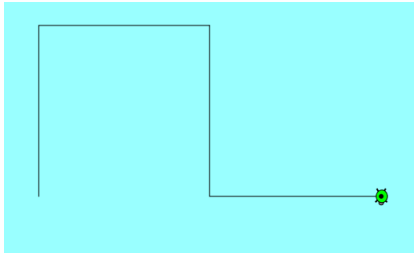
Na Figura 4.4(a) é mostrada a primeira iteração, ou seja, o axioma FX. Na Figura 4.4(b) é descrita a segunda iteração. Por fim, na Figura 4.4(c) detalha-se a terceira iteração.



(a) Tartaruga modelando a primeira iteração, ou seja, o axioma FX.



(b) Segunda iteração da curva do Dragão, ou seja, houve a seguinte substituição: $FX \to FX + YF+$.



(c) Terceira iteração da curva do Dragão, ou seja, houve a seguinte substituição: $FX + YF+ \to FX + YF + + - FX - YF+$.

Figura 4.4: Simulação da primeira à terceira iteração na formação da curva do Dragão.

Continuando o processo iterativo, obtém-se as seguintes iterações. Para mais uma ilustração, são mostradas a seguir a quarta e a quinta iterações:

$FX+YF++-FX-YF++-FX+YF+--FX-YF+;$

$FX+YF++-FX-YF++-FX+YF+--FX-YF++-FX+YF++-FX-YF+--FX+YF+--FX-YF+;$

Veja Figura 4.5 que contém a simulação da 11^a geração da curva do Dragão.

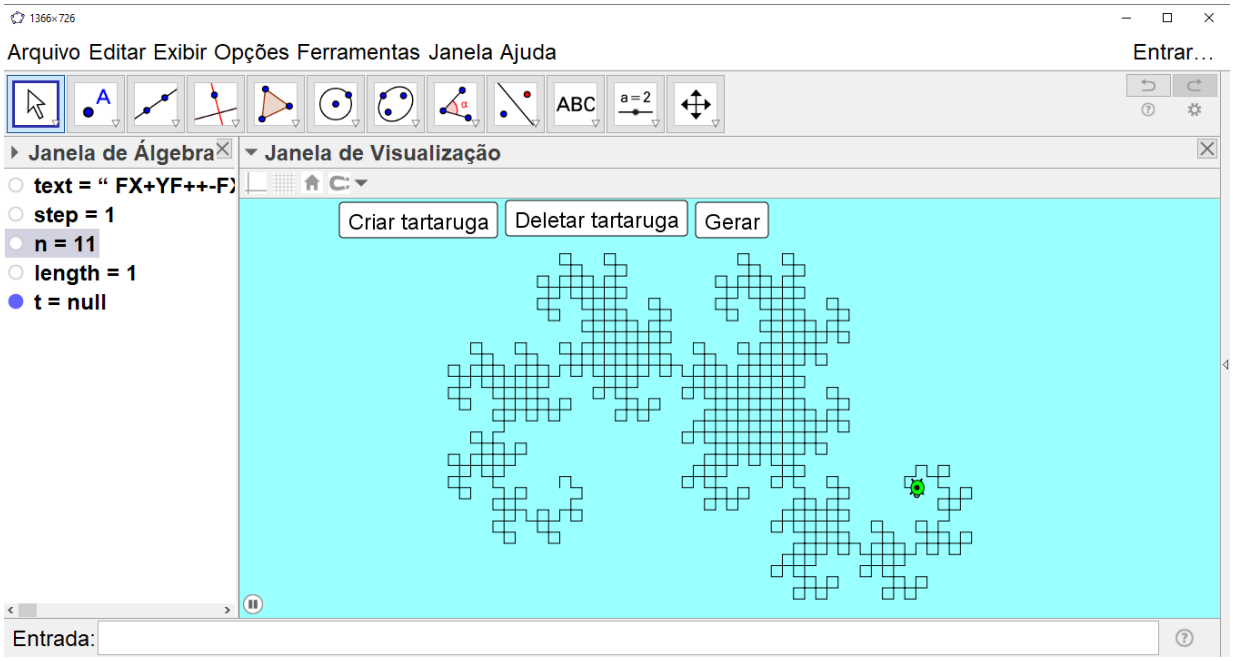


Figura 4.5: Ambiente do GeoGebra com todos os elementos de programação e iteração com usuário. Note que $n = 11$ na "Janela de Álgebra" representa a 11^a geração da curva do Dragão e com passo de tamanho escolhido $length = 1$.

A evolução na pesquisa desta etapa assim como os resultados obtidos de acordo com o exposto na literatura, incentiva a um novo desafio: a realização da modelagem do criador dos L-sistemas, Lindenmayer. Assim, no Capítulo 5 a seguir é desenvolvida a modelagem original das L-árvores axiais que deram fundamento axiomático à teoria dos L-sistemas.

Capítulo 5

Árvores de Lindenmayer

Como observado nos capítulos anteriores associa-se uma frase motivacional relacionada com o assunto principal do capítulo. Dessa forma, começa-se o mais importante capítulo deste trabalho com as palavras dos próprios Lindenmayer e Prusinkiewicz expõe no seu livro:

> Os sistemas de Lindenmayer – ou L-sistemas abreviado – [além de serem muito importantes na geração de fractais] são concebidos como uma teoria matemática do desenvolvimento de plantas [...] o reino vegetal é dominado por estruturas ramificadas; assim, uma descrição matemática de formas semelhantes a árvores e métodos para gerá-los são necessários para fins de modelagem [44].

Neste capítulo, modelam-se árvores de Lindenmayer que foram o fundamento axiomático da teoria dos L-sistemas.

5.1 A Modelagem Botânica de Lindenmayer

São definidos nesta seção alguns conceitos importantes para a compreensão da modelagem matemática e computacional das árvores de Lindenmayer.

Definição 5.1. Uma L-árvore enraizada é uma árvore de Lindenmayer que possui segmentos de ramos que são classificados por níveis, com o extremo inicial em um *nó*. Além disso, todos os segmentos de ramos são construídos na direção do crescimento de uma planta real, ou seja, da raiz ao topo. O nível é classificado em categorias: o primeiro corresponde à base da árvore (raiz e tronco), o segundo nível corresponde às ramificações laterais e, assim, finaliza com o terceiro nível que são as ramificações terminais que determinam os nós terminais do topo da árvore. As sequências de segmentos formam caminhos, a partir de um nó inicial chamado raiz ou base, até um nó terminal (topo da L-árvore). Esta L-árvore é ilustrada na Figura 5.1.

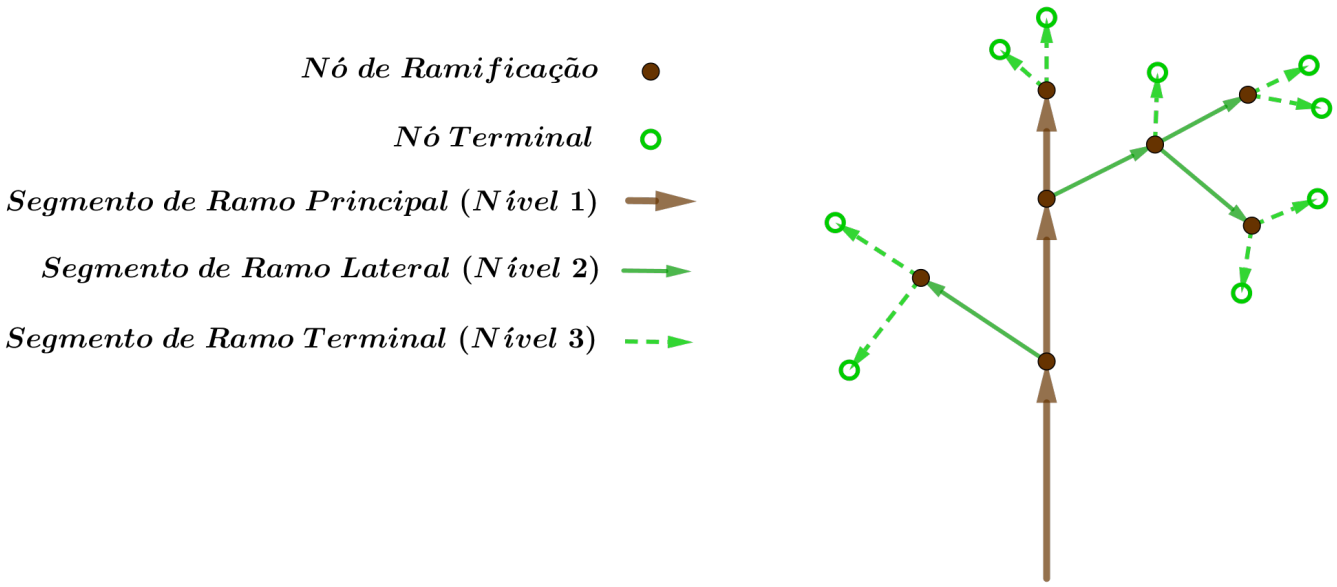


Figura 5.1: O esquema de uma L-árvore enraizada.

Definição 5.2. Uma L-árvore axial é um tipo especial de L-árvore enraizada. Em cada um de seus nós, no máximo um segmento de saída é considerado o principal. Todas as arestas restantes são chamadas de segmentos laterais e terminais. Além disso, uma regra de produção substitui um ramo antecessor por uma L-árvore axial cuja base é o nó inicial do antecessor.

Segundo Lindenmayer e Prusinkiewcz [44], as L-árvores axiais são estruturas que são modeladas utilizando os colchetes $[,]$ que, a partir deste momento, pertencem ao alfabeto V do L-sistema. No caso de L-árvores axiais, estes colchetes, teoricamente, possuem a função de delimitar os nós dos ramos dessas árvores.

Como nos capítulos anteriores, primeiramente mostra-se a estrutura do L-sistema de cada L-árvore, logo após as linhas de código e, assim apresenta-se a simulação final.

5.2 Simulações de Árvores de Lindenmayer

Seja o L-sistema $G = (V, \omega, P)$ dado por:

Alfabeto: $V = \{F, +, -, [,], \{\delta = 10°\}\}$;

Axioma: $\omega = F$;

Regras de Produção: $P = \{F \to FF + [+F - F - F] - [-F + F + F], + \to +, - \to -, [\to [,] \to]\}$.

O L-sistema G representa uma modelagem de uma L-árvore axial. De fato, seu código de programação em *JavaScript*, e sua simulação é exibida a seguir, com uma explicação detalhada das linhas de seu código.

```
var sentenca, x, y, anguloAtual;
var pilhaX = [];
var pilhaY = [];
```

```
var pilhaAngulos = [];
function ggbOnInit() {}

function posicionar() {
    ggbApplet.evalCommand("length=1");
    ggbApplet.evalCommand("T=Turtle[]");
    sentenca = "F";
    anguloAtual = 0;
    anguloArmazenado = 0;
    ggbApplet.setTextValue("text1", "");
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.setTextValue("text1", "");
    ggbApplet.evalCommand("length=0");
}

function movimentacao(sentenca, cont) {
    var atual = sentenca.charAt(cont);
    var correcaoAngulo, contAngulo, anguloArmazenado;
    if (atual == "F") {
        ggbApplet.evalCommand("TurtleForward[T, length]");
    } else if (atual == "+") {
        ggbApplet.evalCommand("TurtleRight[T, angle]");
        anguloAtual = anguloAtual + 1;
    } else if (atual == "-") {
        ggbApplet.evalCommand("TurtleLeft[T, angle]");
        anguloAtual = anguloAtual - 1;
    } else if (atual == "[") {
        x = ggbApplet.getXcoord("T");
        y = ggbApplet.getYcoord("T");
        pilhaX.push(x);
        pilhaY.push(y);
        pilhaAngulos.push(anguloAtual);
    } else if (atual == "]") {
```

```
        x = pilhaX.pop();
        y = pilhaY.pop();
        anguloArmazenado = pilhaAngulos.pop();
        ggbApplet.setCoords("T", x, y);
        correcaoAngulo = anguloArmazenado - anguloAtual;
        if (correcaoAngulo < 0) {
            correcaoAngulo = correcaoAngulo * (-1);
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleLeft[T, angle]");
                contAngulo = contAngulo + 1;
            }
        } else if (correcaoAngulo > 0) {
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleRight[T, angle]");
                contAngulo = contAngulo + 1;
            }
        }
    }
}

function moverTartaruga() {
    ggbApplet.setTextValue("text1", sentenca);
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.evalCommand("T=Turtle[]");
    ggbApplet.evalCommand("TurtleLeft[T, pi/2]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    var proximaSentenca = " ";
    var atual = " ";

    for (var contSent = 0; contSent < sentenca.length; contSent++) {
        movimentacao(sentenca, contSent);
    }

    for (var i = 0; i < sentenca.length; i++) {
```

```
        atual = sentenca.charAt(i);
        if (atual == "F") {
           proximaSentenca = proximaSentenca + "FF+[+F-F-F]-[-F+F+F]";
        } else if ((atual == "+")||(atual == "-")||
                   (atual == "[")||(atual == "]")){
           proximaSentenca = proximaSentenca + atual;
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1")
}
```

De início, tem-se a criação de sete variáveis: $sentenca$, x, y, $anguloAtual$, $pilhaX = []$, $pilhaY = []$, $pilhaAngulos = []$, as quais auxiliam na modelagem. Logo após, foram definidas quatro funções: $posicionar()$, $deletarTartaruga()$, $movimentacao(sentenca, cont)$, $movertartaruga()$, de modo a organizar o programa. Observe que a terceira função definida $movimentacao(sentenca, cont)$, para ser executada, necessita de duas entradas (variáveis) que são: $sentenca$ e $cont$. Os valores dessas duas variáveis são definidos dentro da quarta função, ou seja, para a terceira função ser executada, necessita-se que a quarta seja executada primeiro. Dessa forma, pode-se concluir também que a ordem em que aparecem as funções, no programa, não afetam esta modelagem.

Assim, igual a todas as outras modelagens anteriores, a primeira função $posicionar()$ tem o papel de iniciar a tartaruga e, além disso, prepará-la para realizar o axioma dado por: F. Esta função é realizada ao clicar no botão "Criar tartaruga". Além disso, é criado, também, outras duas variáveis $anguloAtual = 0$ e $anguloArmazenado = 0$ que são utilizadas para corrigir o ângulo armazenado na pilha de ângulos.

A segunda função, $deletarTartaruga()$, deleta toda a modelagem e é ativada quando se clica no botão "Deletar tartaruga".

Neste momento, antes de inicar a descrição detalhada da terceira função $movimentacao(sentenca, cont)$, explicita-se, primeiro, a quarta função $moverTartaruga()$. Esta função é realizada ao clicar no botão "Gerar Árvore". Inicialmente foi definida a seguinte sequência de códigos:

$$
\begin{aligned}
&ggbApplet.evalCommand(\text{“}Delete[T]\text{”}),\\
&ggbApplet.evalCommand(\text{“}TurtleLeft[T, pi/2]\text{”}),\\
&ggbApplet.evalCommand(\text{“}T = Turtle[]\text{”}),\\
&ggbApplet.evalCommand(\text{“}StartAnimation[]\text{”}),
\end{aligned}
$$

nesta mesma ordem e que possuem o papel de, no início de cada iteração, excluir toda a modelagem da geração anterior e criar outra tartaruga para inicar a próxima geração. Além disso proporciona um giro

inicial, da tartaruga, no sentido anti-horário com ângulo $\frac{\pi}{2}$ radianos. Após isso, são definidas mais duas variáveis $proximaSentenca$ e $atual$.

O primeiro $for()$, chama a função $movimentacao(sentenca, cont)$, entretanto, substituindo em $cont$ o contador $contSent$ criado no interior do $for()$. Este comando $for()$ está definido para chamar a função $movimentacao(sentenca, cont)$, até o momento em que o contador $contSent$ for menor que o tamanho da *string* $sentenca$.

O segundo $for()$ realiza as devidas substituições dos elementos do alfabeto com o auxílio da variável $atual$ criada anteriormente, de forma a interpretar símbolo por símbolo da variável $sentenca$. A substituição é armazenada na variável $proximaSentenca$.

Por fim, o conteúdo armazenado em $proximaSentenca$ é armazenado em $sentenca$ tornando-se a *string* da nova geração. E a variável n, que foi definida na aba "Ao Clicar" do botão "Criar tartaruga", é incrementada em um unidade, e tem a função de mostrar qual geração está a modelagem computacional.

Retornando na análise do primeiro $for()$, ao chamar a função $movimentacao(sentenca, cont)$, retorna valores para esta função definida logo acima da função $moverTartaruga()$, ou seja, $sentenca$ é igual a sentença (*string*) da geração atual e $cont$ é igual ao valor de $contSent$.

Analisa-se, agora, a terceira função do programa, denominada $movimentacao(sentenca, cont)$. Esta função possui o objetivo de movimentar a tartaruga, e fazer com que ela corrija sua angulação após aparecer o símbolo "]" da sentença atual. Primeiro, a variável $atual$ armazena o símbolo da $sentenca$ que está na posição $cont$, e mais três variáveis são criadas $correcaoAngulo$, $contAngulo$, $AnguloArmazenado$.

Se o símbolo da variável $atual$ é um F, então a tartaruga anda um passo de tamanho $length$, criado pela função $posicinonar()$ ao clicar no botão "Criar tartaruga".

Se for um $+$, a tartaruga gira para o sentido horário um ângulo fixo dado pela variável $angle$ que foi criada no "Campo de Entrada", que neste caso é $angle = 10^{o}$, e além disso, incrementa a variável $anguloAtual$ em uma unidade.

Se for um $-$, a tartaruga gira para o sentido anti-horário o mesmo ângulo fixo dado pela variável $angle = 10^{o}$, e além disso, reduz a variável $anguloAtual$ em uma unidade.

Se for um [, as variáveis x e y criadas inicialmente armazenam as coordenadas atuais da tartaruga de nome T. Além disso, empilha estas coordenadas nas variáveis $pilhaX$ e $pilhaY$ e empilha o ângulo atual que ela se encontra na variável $pilhaAngulos$, com a função conhecida do ***JavaScript***, $push()$.

Se for um], as variáveis x, y e $anguloArmazenado$ criadas inicialmente resgatam as últimas coordenadas e o ângulo da tartaruga, de nome T, que foram empilhados em $pilhaX$, $pilhaY$ e $pilhaAngulos$, respectivamente, através do comando $pop()$. Logo após, são redefinidas as coordenadas da tartaruga para essas últimas que foram desempilhadas. Para que seu ângulo atual seja corrigido para o ângulo anterior que foi resgatado da pilha, é realizado o seguinte procedimento:

1º Na variável $correcaoAngulo$ armazena a diferença entre o que tem armazenado em $anguloArmazenado$ com $anguloAtual$.

2º Se $correcaoAngulo < 0$, multiplica esta variável por (-1) e a compara com uma outra variável $contAngulo = 0$. Enquanto $contAngulo < correcaoAngulo$, a tartaruga realiza giros no sentido anti-horário de ângulo $angle = 10^{o}$. Além disso, incrementa uma unidade em $contAngulo$.

3º Se $correcaoAngulo > 0$, compara, da mesma forma que no 2º item, com a variável $contAngulo = 0$. Enquanto $contAngulo < correcaoAngulo$, a tartaruga realiza giros no sentido horário de ângulo $angle = 10^{o}$. Além disso, incrementa uma unidade em $contAngulo$.

Desta forma, o último ângulo empilhado que a tartaruga se encontrava é resgatado.

Portanto, em suma, sempre que o símbolo [é analisado na *string* $sentenca$, o programa armazena a posição e a angulação da tartaruga com relação ao eixo y, pois de início, foi definido na função $moverTartaruga()$ para a tartaruga sempre realizar um giro de $\frac{\pi}{2}$ radianos no sentido anti-horário. E quando o primeiro símbolo] for analisado na *string* $sentenca$, o programa resgata a posição e a angulação da tartaruga que foram armazenadas no último símbolo [. Veja Figura 5.2 para ilustração do que foi dito anteriormente com relação aos colchetes.



Figura 5.2: O colchete [armazena e empilha uma posição e angulação da tartaruga. Estes dados só serão desempilhados quando chegar no colchete]. O processo é análogo aos colchetes [ ] e [ ] .

Portanto, após a análise do código, veja a Figura 5.3 da simulação desta árvore de Lindenmayer.

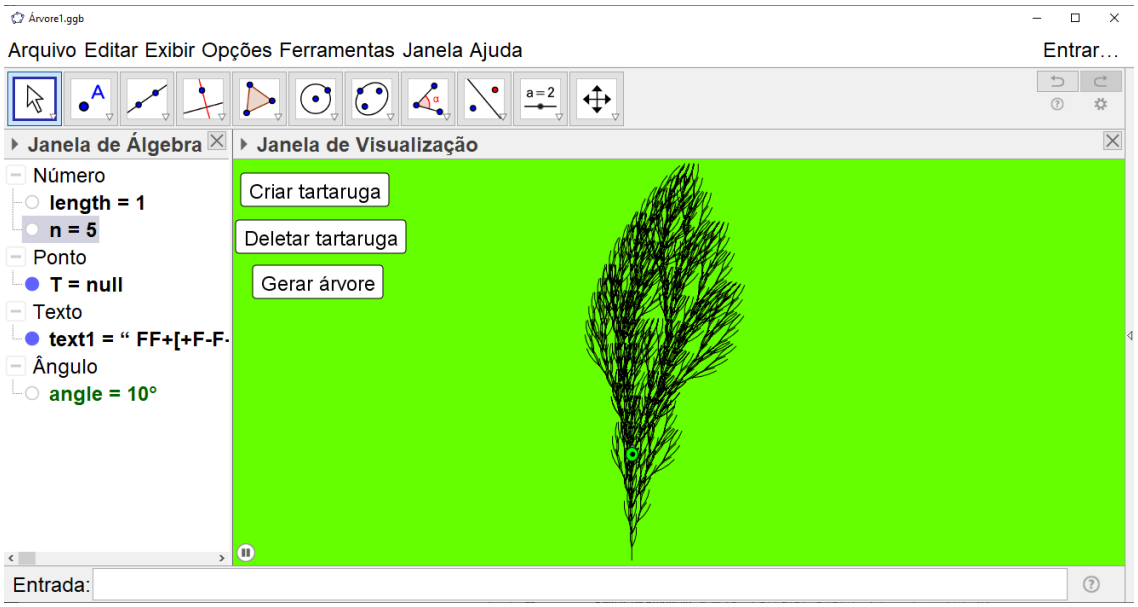


Figura 5.3: Interface do GeoGebra com a modelagem da quinta geração ($n = 5$ na "Janela de Álgebra") de uma árvore de Lindenmayer.

A partir deste código, consegue-se definir outras estruturas de plantas de Lindenmayer, através dos L-sistemas, com outros axiomas e regras de produção. Veja alguns exemplos a seguir.

Seja o L-sistema $G = (V, \omega, P)$, em que:

Alfabeto: $V = \{F, +, -, [,], \{\delta = 5°\}\}$;

Axioma: $\omega = F$;

Regras de Produção: $P = \{F \to F[+F]F[-F][F], + \to +, - \to -, [\to [,] \to]\}$.

Seu código de programação em *JavaScript* é dado a seguir:

```
var sentenca, x, y, anguloAtual;
var pilhaX = [];
var pilhaY = [];
var pilhaAngulos = [];

function ggbOnInit() {}

function posicionar() {
    ggbApplet.evalCommand("length=1");
    ggbApplet.evalCommand("T=Turtle[]");
    sentenca = "F";
    anguloAtual = 0;
    anguloArmazenado = 0;
    ggbApplet.setTextValue("text1", "");
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.setTextValue("text1", "");
    ggbApplet.evalCommand("length=0");
}

function movimentacao(sentenca, cont) {
    var atual = sentenca.charAt(cont);
    var correcaoAngulo, contAngulo, anguloArmazenado;
    if (atual == "F") {
        ggbApplet.evalCommand("TurtleForward[T, length]");
    } else if (atual == "+") {
        ggbApplet.evalCommand("TurtleRight[T, angle]");
```

```
        anguloAtual = anguloAtual + 1;
    } else if (atual == "-") {
        ggbApplet.evalCommand("TurtleLeft[T, angle]");
        anguloAtual = anguloAtual - 1;
    } else if (atual == "[") {
        x = ggbApplet.getXcoord("T");
        y = ggbApplet.getYcoord("T");
        pilhaX.push(x);
        pilhaY.push(y);
        pilhaAngulos.push(anguloAtual);
    } else if (atual == "]") {
        x = pilhaX.pop();
        y = pilhaY.pop();
        anguloArmazenado = pilhaAngulos.pop();
        ggbApplet.setCoords("T", x, y);
        correcaoAngulo = anguloArmazenado - anguloAtual;
        if (correcaoAngulo < 0) {
            correcaoAngulo = correcaoAngulo * (-1);
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleLeft[T, angle]");
                contAngulo = contAngulo + 1;
            }
        } else if (correcaoAngulo > 0) {
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleRight[T, angle]");
                contAngulo = contAngulo + 1;
            }
        }
    }
}

function moverTartaruga() {
    ggbApplet.setTextValue("text1", sentenca);
    ggbApplet.evalCommand("Delete[T]");
```

```
    ggbApplet.evalCommand("T=Turtle[]");
    ggbApplet.evalCommand("TurtleLeft[T, pi/2]");
    ggbApplet.evalCommand("StartAnimation[]");
    var proximaSentenca = " ";
    var atual = " ";
    for (var contSent = 0; contSent < sentenca.length; contSent++) {
        movimentacao(sentenca, contSent);
    }
    for (var i = 0; i < sentenca.length; i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            proximaSentenca = proximaSentenca + "F[+F]F[-F][F]";
        } else if ((atual == "+") || (atual == "-") ||
                   (atual == "[") || (atual == "]")) {
            proximaSentenca = proximaSentenca + atual;
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
}
```

Veja Figura 5.4 para visualização da simulação obtida.

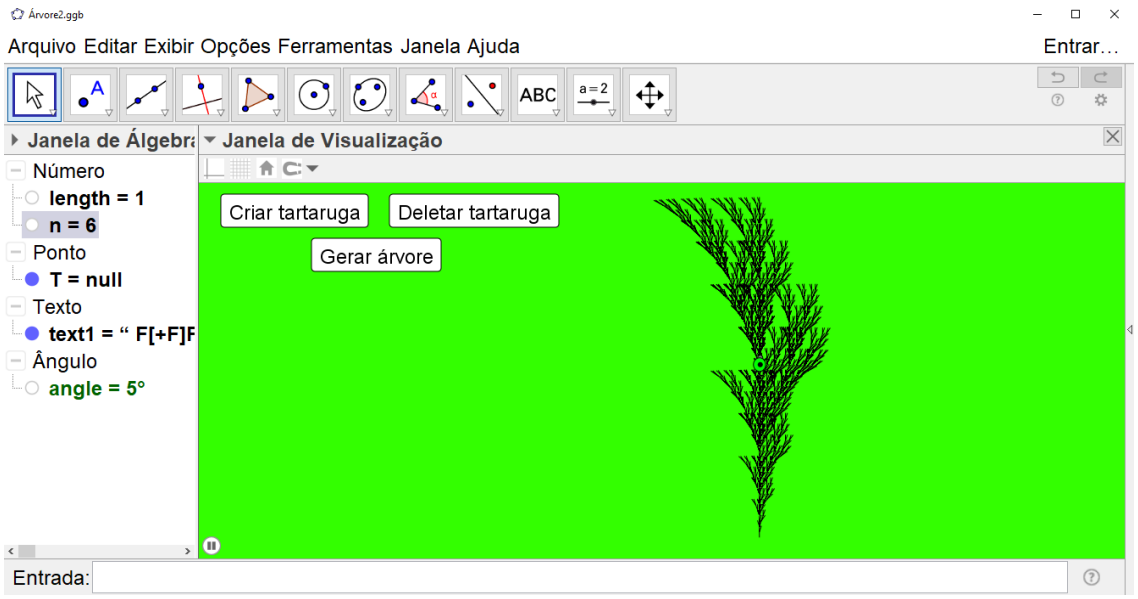


Figura 5.4: Interface do GeoGebra com a modelagem da sexta geração ($n = 6$ na "Janela de Álgebra") de outra árvore de Lindenmayer.

Para finalizar, apresenta-se um último exemplo de uma L-árvore axial de Lindenmayer.

Seja o L-sistema $G = (V, \omega, P)$, em que:

Alfabeto: $V = \{F, +, -, [,], \{\delta = 7^\circ\}\}$;

Axioma: $\omega = F$;

Regras de Produção: $P = \{F \to F[+F]F[-F]F, + \to +, - \to -, [\to [,] \to]\}$.

Seu código de programação em *JavaScript* é dado a seguir:

```
var sentenca, x, y, anguloAtual;
var pilhaX = [];
var pilhaY = [];
var pilhaAngulos = [];
function ggbOnInit() {}

function posicionar() {
    ggbApplet.evalCommand("length=1");
    ggbApplet.evalCommand("T=Turtle[]");
    sentenca = "F";
    anguloAtual = 0;
    anguloArmazenado = 0;
    ggbApplet.setTextValue("text1", "");
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.setTextValue("text1", "");
    ggbApplet.evalCommand("length=0");
}

function movimentacao(sentenca, cont) {
    var atual = sentenca.charAt(cont);
    var correcaoAngulo, contAngulo, anguloArmazenado;
    if (atual == "F") {
        ggbApplet.evalCommand("TurtleForward[T, length]");
    } else if (atual == "+") {
        ggbApplet.evalCommand("TurtleRight[T, angle]");
        anguloAtual = anguloAtual + 1;
    } else if (atual == "-") {
        ggbApplet.evalCommand("TurtleLeft[T, angle]");
```

```
        anguloAtual = anguloAtual - 1;
    } else if (atual == "[") {
        x = ggbApplet.getXcoord("T");
        y = ggbApplet.getYcoord("T");
        pilhaX.push(x);
        pilhaY.push(y);
        pilhaAngulos.push(anguloAtual);
    } else if (atual == "]") {
        x = pilhaX.pop();
        y = pilhaY.pop();
        anguloArmazenado = pilhaAngulos.pop();
        ggbApplet.setCoords("T", x, y);
        correcaoAngulo = anguloArmazenado - anguloAtual;
        if (correcaoAngulo < 0) {
            correcaoAngulo = correcaoAngulo * (-1);
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleLeft[T, angle]");
                contAngulo = contAngulo + 1;
            }
        } else if (correcaoAngulo > 0) {
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleRight[T, angle]");
                contAngulo = contAngulo + 1;
            }
        }
    }
}

function moverTartaruga() {
    ggbApplet.setTextValue("text1", sentenca);
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.evalCommand("T=Turtle[]");
    ggbApplet.evalCommand("TurtleLeft[T, pi/2]");
    ggbApplet.evalCommand("StartAnimation[]");
```

```
    var proximaSentenca = " ";
    var atual = " ";
    for (var contSent = 0; contSent < sentenca.length; contSent++) {
        movimentacao(sentenca, contSent);
    }
    for (var i = 0; i < sentenca.length; i++) {
        atual = sentenca.charAt(i);
        if (atual == "F") {
            proximaSentenca = proximaSentenca + "F[+F]F[-F]F";
        } else if ((atual == "+") || (atual == "-") ||
                   (atual == "[") || (atual == "]")) {
            proximaSentenca = proximaSentenca + atual;
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
}
```

Veja Figura 5.5 para visualização da simulação obtida.

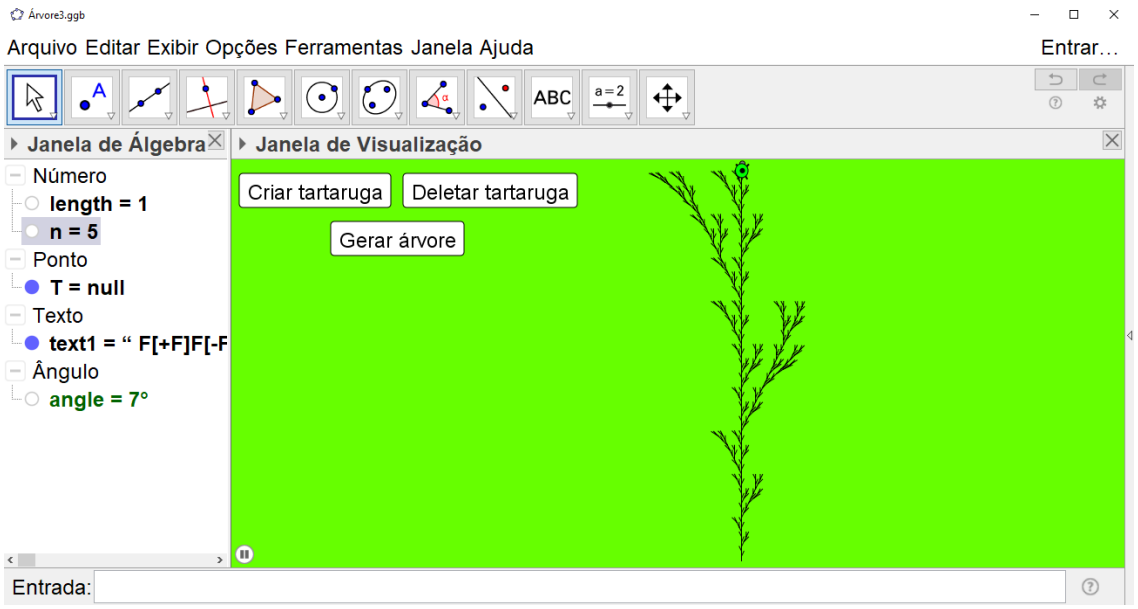


Figura 5.5: Interface do GeoGebra com a modelagem da quinta geração ($n = 5$ na "Janela de Álgebra") de outra árvore de Lindenmayer.

Como um dos grandes desafios desta investigação chega a sua culminância em forma sucedida, a curiosidade de aplicar os axiomáticos da teoria dos L-sistemas encontra um novo desafio: a tesselação de Roger Penrose. Uma nova etapa é iniciada e, como toda iniciação, precisa-se de uma retomada às pesquisas que levam à fundamentação do novo tópico a ser abordado. No Capítulo 6 seguinte é ilustrada essa caminhada.

Capítulo 6

Tesselações via L-sistemas

"Não tenha medo de pular equações (eu mesmo faço isso com frequência)."
Roger Penrose, Prefácio de The Road to Reality [42], (Londres, 2005).

6.1 Tesselações: Alicerce Teórico

A arte de representar tesselações, surgiram desde as civilizações e tempos antigos. Conceitos associados à tesselação como pavimentação e mosaico possuem na literatura matemática o mesmo significado, como apontam Grünbaum e Shephard [16]. Juntamente com a noção de padronização, isto é, processos (neste caso, figuras) que se repetem em uma maneira sistemática [16], as tesselações tornaram-se esteticamente mais agradáveis e harmoniosas aos olhos. Romanos e povos mediterrâneos utilizavam representações de seres humanos e elementos da natureza para construir seus mosaicos. Mouros e árabes utilizavam formatos e cores de pavimentos para retratar figuras geométricas complexas. Na Idade Média, as pavimentações serviam para recobrir chãos, tetos e paredes. E a fim de exemplificar, nos tempos contemporâneos, a padronização e as tesselações são encontradas em diversas áreas, como nas engenharias e na arquitetura [31], ou na vida corriqueira é muito comum encontrar mosaicos nos azuleijos da cozinha e dos banheiros, nos pisos e nos terraços das nossas casas.

Johannes Kepler, famoso astrônomo, astrólogo e matemático, que viveu em meados dos séculos XVI-XVII, foi o primeiro cientista a dar um tratamento rigoroso matemático para as tesselações. Em seu livro *Harmonices Mundi* [26], publicado em 1619, Kepler além de fornecer imagens de mosaicos, também realizou contribuições indispensáveis para a teoria das tesselações.

Segue uma definição no rigor matemático para o conceito de tesselação. Para isto, precisa-se do conceito de interior topológico de um conjunto em $\mathbb{R}^2$. Seja $A \subset \mathbb{R}^2$, o interior topológico de A, denotado por $\overset{\circ}{A}$ é o conjunto dos pontos de A tais que uma bola aberta centrada num ponto está inteiramente contida em A [29].

Definição 6.1. Seja T uma família de conjuntos fechados no plano $\mathbb{R}^2$, $T = \{T_\epsilon\}_{\epsilon \in L}$, em que L é um conjunto de índices arbitrários, tais que:

$$\text{(a)} \bigcup_{\epsilon \in L} T_\epsilon = \mathbb{R}^2, \qquad \text{(b)} \overset{\circ}{T}_{\epsilon_i} \cap \overset{\circ}{T}_{\epsilon_j} = \emptyset, \text{ para todo } \epsilon_i, \epsilon_j \in L,\ \epsilon_i \neq \epsilon_j.$$

Dessa forma, diz-se que T é uma tesselação do plano $\mathbb{R}^2$.

Os elementos T_ϵ são chamados de pavimentos da tesselação T. Note que parte da fronteira topológica de T_ϵ [29] pode ser compartilhada por outros elementos da família T.

Neste estudo, os conjuntos fechados T_ϵ são figuras geométricas, em particular polígonos, como os apresentados no Capítulo 1. Lembre-se que nas tesselações do Capítulo 1 utiliza-se polígonos regulares congruentes para determinar a tesselação. De fato, quando todos os elementos da família $T = \{T_\epsilon\}_{\epsilon \in L}$ são congruentes a uma única figura F, a tesselação T é chamada de tesselação unicelular gerada por F. Se F é um polígono regular, a tesselação é chamada unicelular regular.

Nas próximas seções, introduz-se uma tesselação construída com a dinâmica dos L-sistemas e as dificuldades e limitações encontradas para modelagem da tesselação de Penrose através da tartaruga do GeoGebra.

6.2 Uma Primeira Tesselação Utilizando um L-sistema

A fim de mesclar o conceito de tesselação deste capítulo com o de L-sistemas 2, apresenta-se a modelagem de uma tesselação, que possui dois pavimentos geradores: um hexágono regular e um losango. Este losango é congruente a um pavimento da tesselação de Penrose apresentada na Seção 6.3.

Seja o L-sistema $G = (V, \omega, P)$, em que:

Alfabeto: $V = \{F, +, -, X, Y, Z, W, \{\delta = 30°\}\}$;

Axioma: $\omega = W$;

Regras de Produção: $P = \{W \to +++X--F--ZFX+, X \to ---W++F++YFW-, Y \to +ZFX--F--Z+++, Z \to -YFW++F++Y---, + \to +, - \to -\}$.

Seu código de programação em *JavaScript* é dado a seguir:

```
var sentenca, x, y, anguloAtual;
var pilhaX = [];
var pilhaY = [];
var pilhaAngulos = [];
var X;
var Y;
var Z;
var W;

function ggbOnInit() {}

function posicionar() {
```

```
    ggbApplet.evalCommand("length=1");
    ggbApplet.evalCommand("T=Turtle[]");
    sentenca = "W";
    anguloAtual = 0;
    anguloArmazenado = 0;
    ggbApplet.setTextValue("text1", "");
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.setTextValue("text1", "");
    ggbApplet.evalCommand("length=0");
}

function movimentacao(sentenca, cont) {
    var atual = sentenca.charAt(cont);
    var correcaoAngulo, contAngulo, anguloArmazenado;
    if (atual == "F") {
        ggbApplet.evalCommand("TurtleForward[T, length]");
    } else if (atual == "+") {
        ggbApplet.evalCommand("TurtleRight[T, angle]");
        anguloAtual = anguloAtual + 1;
    } else if (atual == "-") {
        ggbApplet.evalCommand("TurtleLeft[T, angle]");
        anguloAtual = anguloAtual - 1;
    } else if (atual == "[") {
        x = ggbApplet.getXcoord("T");
        y = ggbApplet.getYcoord("T");
        pilhaX.push(x);
        pilhaY.push(y);
        pilhaAngulos.push(anguloAtual);
    } else if (atual == "]") {
        x = pilhaX.pop();
        y = pilhaY.pop();
        anguloArmazenado = pilhaAngulos.pop();
        ggbApplet.setCoords("T", x, y);
```

```
        correcaoAngulo = anguloArmazenado - anguloAtual;
        if (correcaoAngulo < 0) {
            correcaoAngulo = correcaoAngulo * (-1);
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleLeft[T, angle]");
                contAngulo = contAngulo + 1;
            }
        } else if (correcaoAngulo > 0) {
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleRight[T, angle]");
                contAngulo = contAngulo + 1;
            }
        }
    }
}

function moverTartaruga() {
    ggbApplet.setTextValue("text1", sentenca);
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.evalCommand("T=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    var proximaSentenca = " ";
    var atual = " ";
    for (var contSent = 0; contSent < sentenca.length; contSent++) {
        movimentacao(sentenca, contSent);
    }
    for (var i = 0; i < sentenca.length; i++) {
        atual = sentenca.charAt(i);
        if (atual == "W") {
            proximaSentenca = proximaSentenca + "+++X--F--ZFX+"
        } else if (atual == "X") {
            proximaSentenca = proximaSentenca + "---W++F++YFW-"
        } else if (atual == "Y") {
            proximaSentenca = proximaSentenca + "+ZFX--F--Z+++"
```

```
        } else if (atual == "Z") {
            proximaSentenca = proximaSentenca + "-YFW++F++Y---"
        } else if ((atual == "+") || (atual == "-") || (atual == "F") {
            proximaSentenca = proximaSentenca + atual;
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1")
}
```

A interpretação desta codificação é análoga às interpretações do Capítulo 5, com exceção de que nesta modelagem há ausência de colchetes. Ou seja, a dinâmica não empilha e nem desempilha informações acerca das coordenadas e angulação da tartaruga. Além disso, são utilizadas quatro variáveis para controle das iterações e que são substituídas a cada nova geração, da mesma forma que ocorre com os L-sistemas do Capítulo 4.

Na Figura 6.1 ilustra-se a modelagem computacional da tesselação relacionada ao código de *JavaScript* anterior.

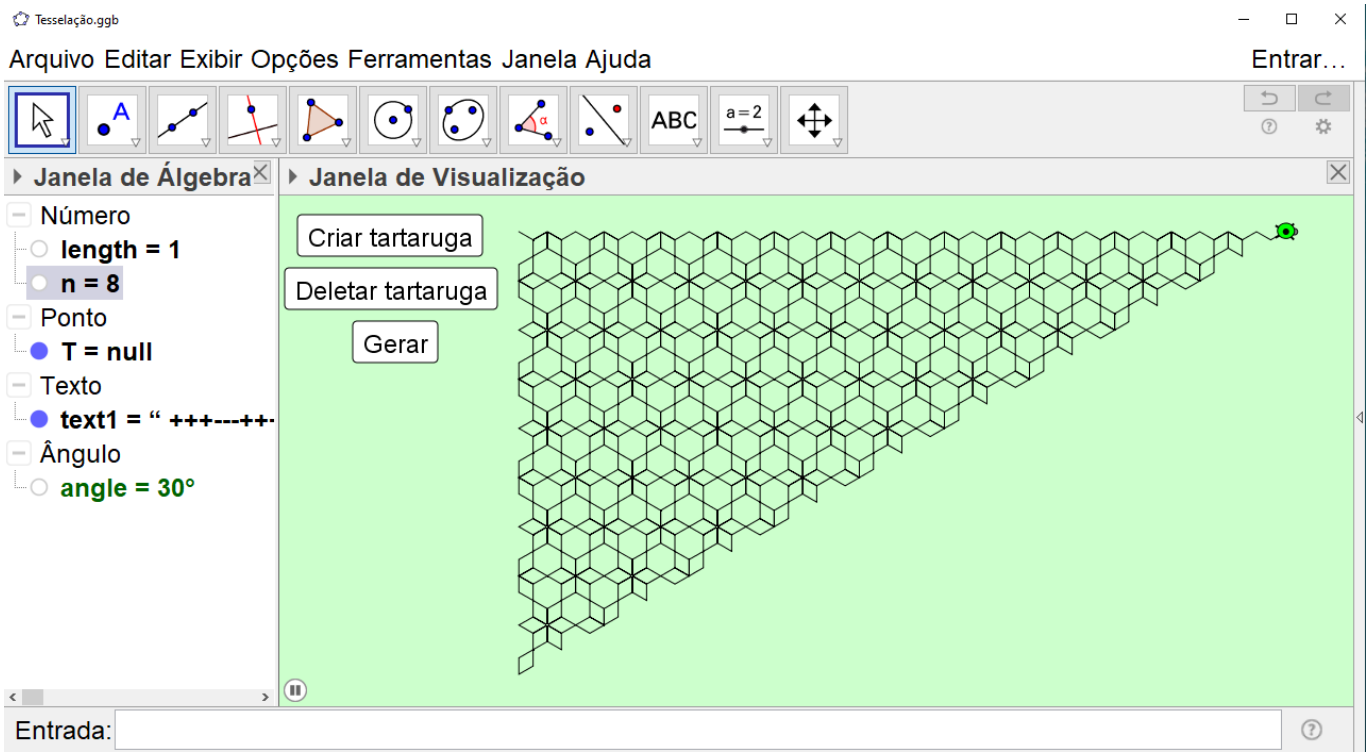


Figura 6.1: Interface do GeoGebra com a modelagem de uma tesselação da oitava geração ($n = 8$ na "Janela de Álgebra").

6.3 Esbarrando com o Primeiro Problema Técnico: A Tesselação de Penrose

Sir Roger Penrose, nascido em 1931, é um físico matemático, filósofo da ciência e ganhador do Prêmio Nobel de Física 2020. Professor Emérito *Rouse Ball* de Matemática da Universidade de Oxford, Inglaterra, com uma imensa contribuição científica na literatura ([39–41]).

Uma tesselação de Penrose tem muitas propriedades notáveis, dentre as mais destacadas são:

- Não é periódica, o que significa que não tem simetria translacional. Mais informalmente, uma mudança em uma parte não corresponde exatamente à parte original.
- Qualquer região finita de uma tesselação de Penrose aparece infinitamente na própria tesselação. Esta propriedade seria trivialmente verdadeira para uma tesselação com simetria translacional, mas não é trivial quando aplicado às tesselações de Penrose que são aperiódicas.
- É um quasicristal [46].

Falando em simetria, com as próprias palavras de Roger Penrose:

> A noção de simetria tem manifestações amplamente difundidas na cultura humana. A simetria também exibe outras interessantes características do funcionamento do mundo natural. É uma noção simples o suficiente para ser entendido por um jovem ou uma criança; no entanto, é sutil o suficiente para ser o centro do mais profundo e mais bem-sucedidas teorias físicas que descrevem o funcionamento interno da natureza. Simetria é, portanto, um conceito que é simultaneamente óbvio e profundo [50].



Figura 6.2: Roger Penrose no saguão do *Mitchell Institute for Fundamental Physics and Astronomy, Texas A&M University*, em pé no chão com a tesselação de Penrose modelada nesta seção.

Fonte: Retirado da *Wikipedia*.

Nesta seção é apresentada a tesselação de Penrose da Figura 6.2. utilizando um L-sistema $G_1 = (V, \omega, P)$ em que:

Alfabeto: $V = \{F, +, -, X, Y, W, Z, [,], \{\delta = 36°\}\}$;

Axioma: $\omega_1 = [X] + +[X] + +[X] + +[X] + +[X]$;

Regras de Produção: $P = \{W \to YF + +ZF - - - -XF[-YF - - - -WF] + +, X \to +YF - -ZF[- - -WF - -XF] +, Y \to -WF + +XF[+ + +YF + +ZF] -, Z \to - -YF + + + +WF[+ZF + + + + +XF] - -XF, + \to +, - \to -, [\to [,] \to]\}$.

Nas primeiras simulações da modelagem foi utilizado o mesmo código das L-árvores axiais as quais utilizam os colchetes no alfabeto. Todavia, observou-se erros desde a segunda geração, em que também foram realizados testes de mesa. A justificativa encontrada para isso era a formulação do L-sistema G_1. Donde decidiu-se substituir o axioma ω_1 pelo $\omega_2 = +WF - -XF - - - YF - -ZF$. Com o L-sistema $G_2 = (V, \omega_2, P)$ cujo código, denominado Código 1 que se apresenta a seguir, as três primeiras gerações ocorreram como esperado.

```
var sentenca, x, y, anguloAtual;
var pilhaX = [];
var pilhaY = [];
var pilhaAngulos = [];
var X;
var Y;
var Z;
var W;

function ggbOnInit() {}

function posicionar() {
    ggbApplet.evalCommand("length=1");
    ggbApplet.evalCommand("T=Turtle[]");
    sentenca = "+WF--XF---YF--ZF";
    anguloAtual = 0;
    anguloArmazenado = 0;
    ggbApplet.setTextValue("text1", "");
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.setTextValue("text1", "");
    ggbApplet.evalCommand("length=0");
```

```
}

function movimentacao(sentenca, cont) {
    var atual = sentenca.charAt(cont);
    var correcaoAngulo, contAngulo, anguloArmazenado;
    if (atual == "F") {
        ggbApplet.evalCommand("TurtleForward[T, length]");
    } else if (atual == "-") {
        ggbApplet.evalCommand("TurtleRight[T, angle]");
        anguloAtual = anguloAtual - 1;
    } else if (atual == "+") {
        ggbApplet.evalCommand("TurtleLeft[T, angle]");
        anguloAtual = anguloAtual + 1;
    } else if (atual == "[") {
        x = ggbApplet.getXcoord("T");
        y = ggbApplet.getYcoord("T");
        pilhaX.push(x);
        pilhaY.push(y);
        pilhaAngulos.push(anguloAtual);
    } else if (atual == "]") {
        x = pilhaX.pop();
        y = pilhaY.pop();
        anguloArmazenado = pilhaAngulos.pop();
        ggbApplet.setCoords("T", x, y);
        correcaoAngulo = anguloArmazenado - anguloAtual;
        if (correcaoAngulo < 0) {
            correcaoAngulo = correcaoAngulo * (-1);
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleLeft[T, angle]");
                contAngulo = contAngulo + 1;
            }
        } else if (correcaoAngulo > 0) {
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleRight[T, angle]");
                contAngulo = contAngulo + 1;
            }
```

```
        }
    }
}

function moverTartaruga() {
    ggbApplet.setTextValue("text1", sentenca);
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.evalCommand("T=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    var proximaSentenca = " ";
    var atual = " ";

    for (var contSent = 0; contSent < sentenca.length; contSent++) {
        movimentacao(sentenca, contSent);
    }

    for (var i = 0; i < sentenca.length; i++) {
        atual = sentenca.charAt(i);
        if (atual == "W") {
            proximaSentenca = proximaSentenca + "YF++ZF----XF[-YF----WF]++"
        } else if (atual == "X") {
            proximaSentenca = proximaSentenca + "+YF--ZF[---WF--XF]+"
        } else if (atual == "Y") {
            proximaSentenca = proximaSentenca + "-WF++XF[+++YF++ZF]-"
        } else if (atual == "Z") {
            proximaSentenca = proximaSentenca + "--YF++++WF[+ZF++++XF]--XF";
        } else if ((atual == "+") || (atual == "-") ||  (atual == "[") ||
                   (atual == "]")) {
            proximaSentenca = proximaSentenca + atual;
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
}
```

A partir da quarta geração, que está representada na Figura 6.6, a tartaruga do GeoGebra não segue a dinâmica da modelagem observada em [16]. Além disso notam-se as interseções dos dois losangos que geram a tesselação.

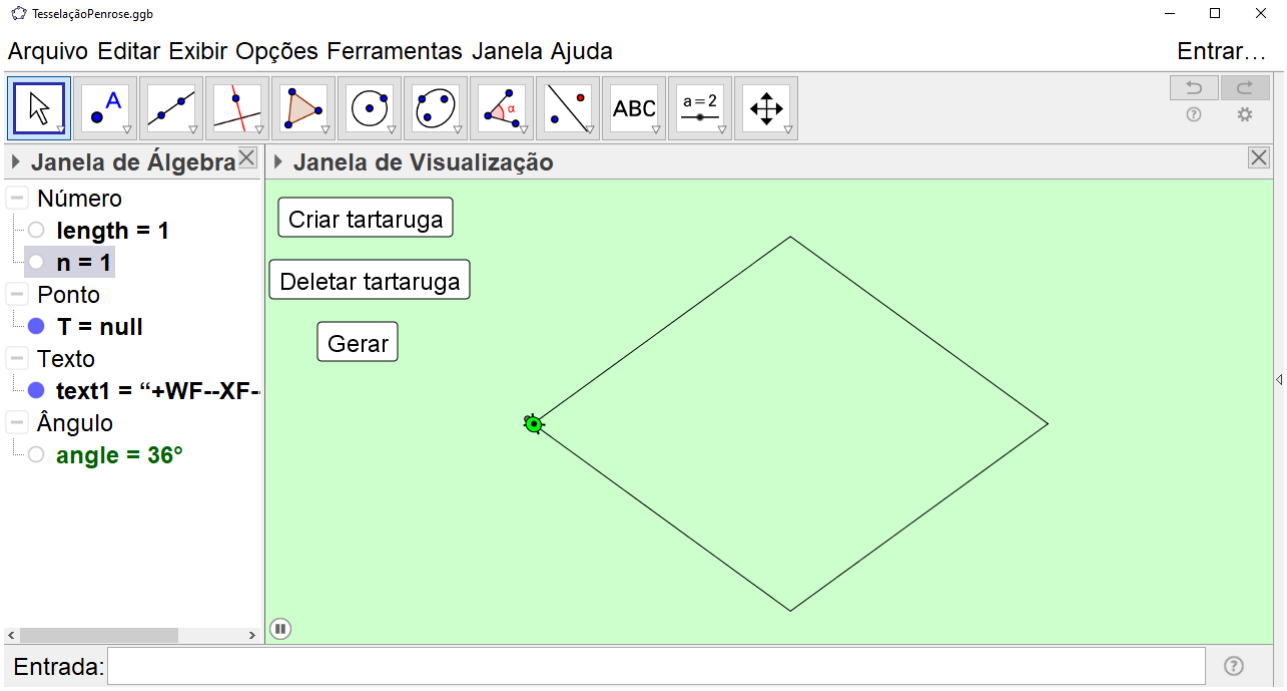


Figura 6.3: Interface do GeoGebra com a modelagem da tesselação de Penrose da primeira geração ($n = 1$ na "Janela de Álgebra").

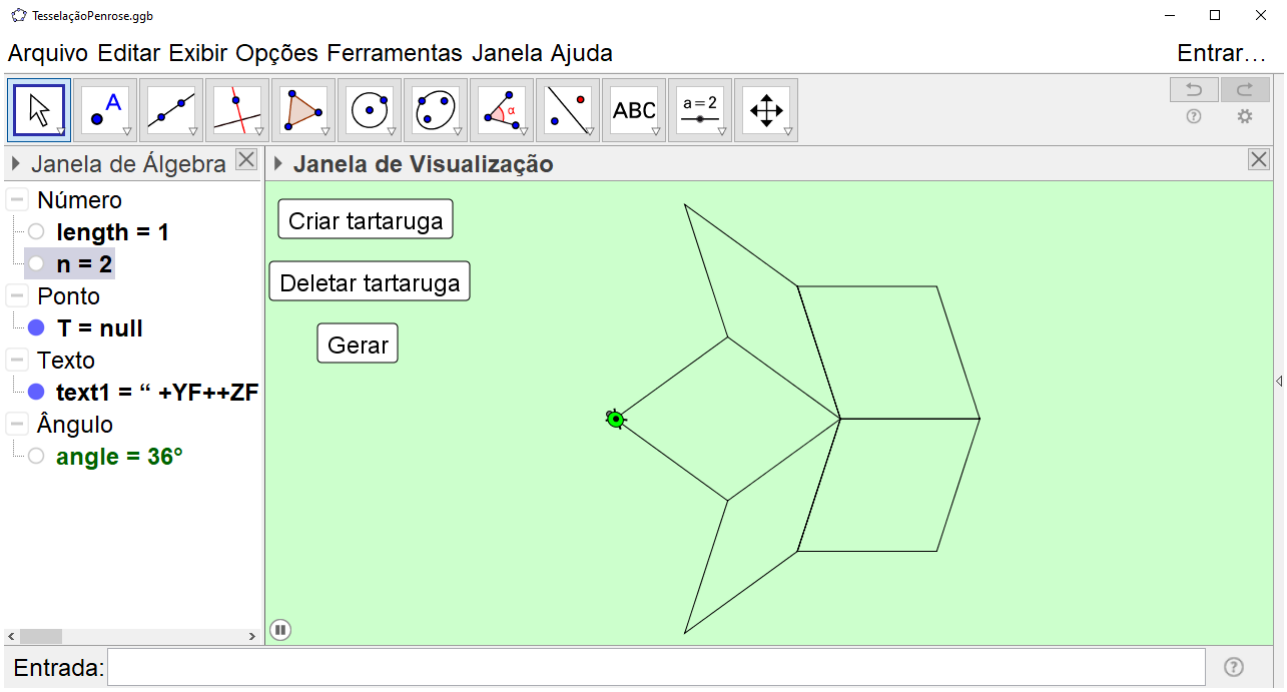


Figura 6.4: Interface do GeoGebra com a modelagem da tesselação de Penrose da segunda geração ($n = 2$ na "Janela de Álgebra").

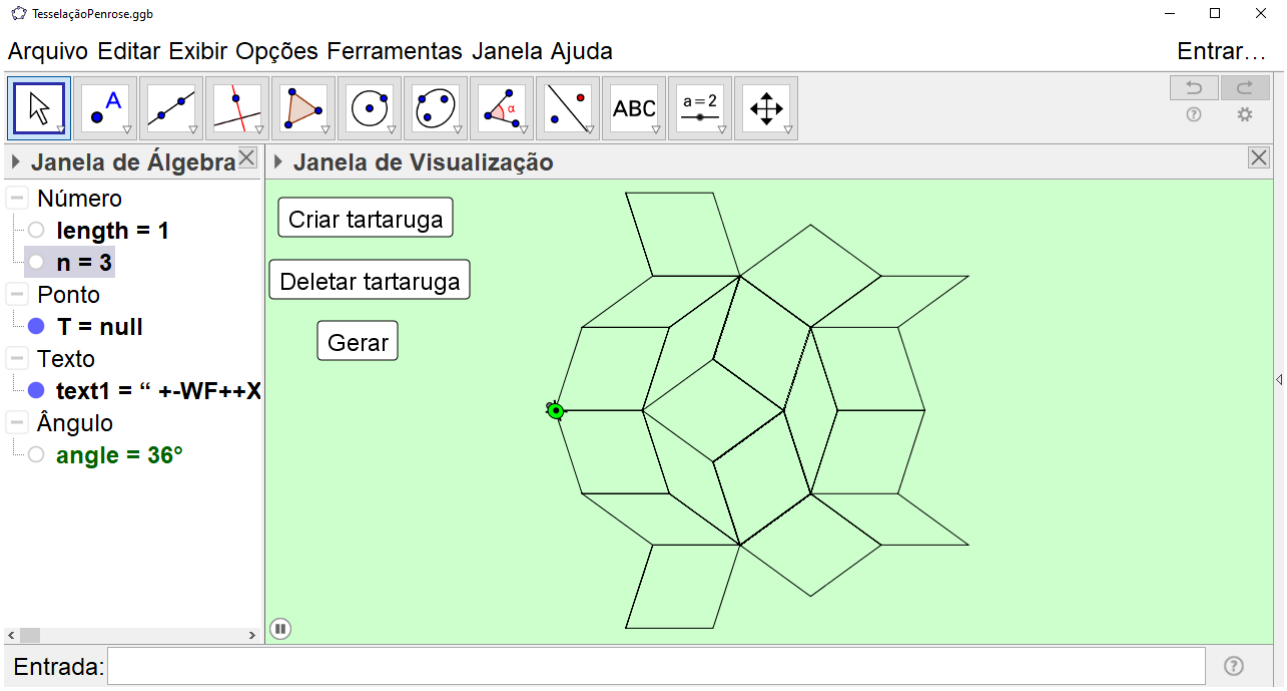


Figura 6.5: Interface do GeoGebra com a modelagem da tesselação de Penrose da terceira geração ($n = 3$ na "Janela de Álgebra").

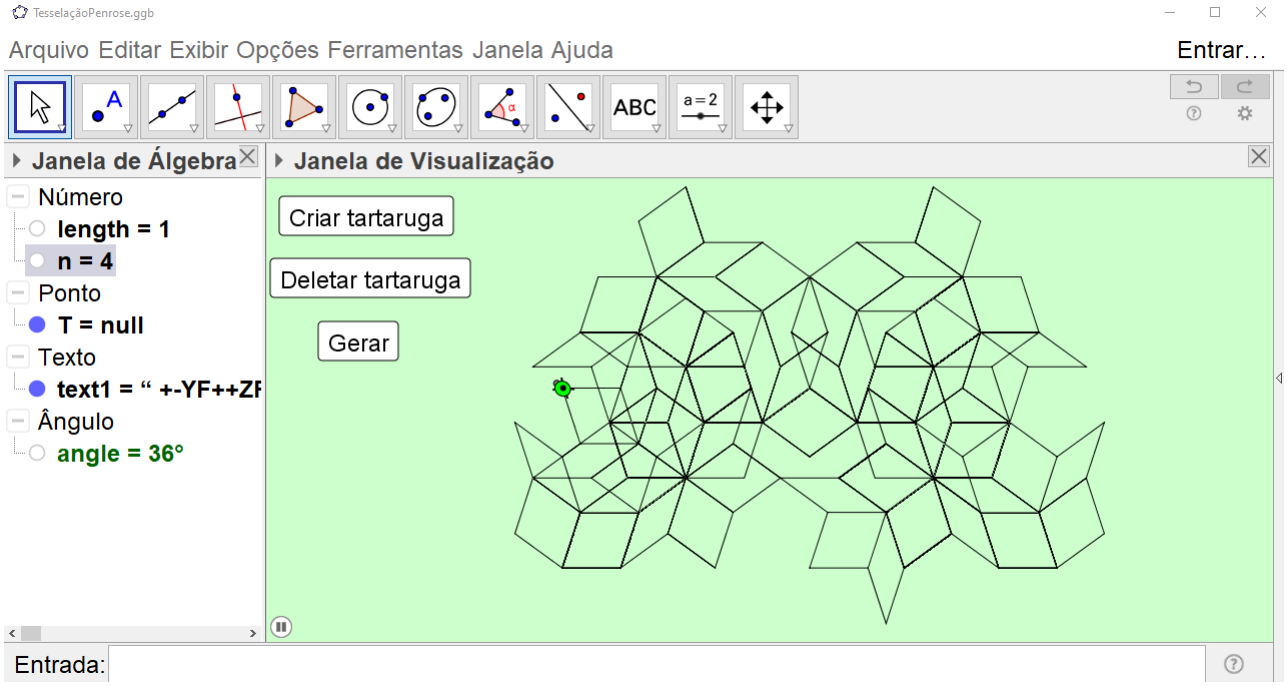


Figura 6.6: Interface do GeoGebra com a modelagem da tesselação de Penrose da quarta geração ($n = 4$ na "Janela de Álgebra").

A primeira conclusão foi a aparente limitação computacional do GeoGebra, desde que a *string* obtida pelas substituições possui já na quarta geração, muitos caracteres. A segunda conclusão foi que a codificação falhava em corrigir os ângulos ângulos ao realizar os colchetes do L-sistema. Para sanar este problema, foram realizados diversas modificações e testes no Código 1 que resultou no Código 2 explicitado a seguir. Note que o L-sistema utilizado é original G_1.

```
var sentenca, x, y, anguloAtual;
var pilhaX = [];
var pilhaY = [];
var pilhaAngulos = [];
var X;
var Y;
var Z;
var W;

function ggbOnInit() {}

function posicionar() {
    ggbApplet.evalCommand("length=1");
    ggbApplet.evalCommand("T=Turtle[]");
    sentenca = "[X]++[X]++[X]++[X]++[X]";
    anguloAtual = 0;
    anguloArmazenado = 0;
    ggbApplet.setTextValue("text1", "");
}

function deletarTartaruga() {
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.setTextValue("text1", "");
    ggbApplet.evalCommand("length=0");
}

function sentencaMovimentacao() {
    ggbApplet.evalCommand("Delete[T]");
    ggbApplet.evalCommand("T=Turtle[]");
    ggbApplet.evalCommand("StartAnimation[ ]");
    var proximaSentenca = "";
    var atual = "";
    for (var i = 0; i < sentenca.length; i++) {
        atual = sentenca.charAt(i);
        if (atual == "X") {
            proximaSentenca = proximaSentenca + "+YF--ZF[---WF--XF]+";
        } else if (atual == "Y") {
            proximaSentenca = proximaSentenca + "-WF++XF[+++YF++ZF]-";
```

```
        } else if (atual == "W") {
            proximaSentenca = proximaSentenca + "YF++ZF----XF[-YF----WF]++";
        } else if (atual == "Z") {
            proximaSentenca = proximaSentenca + "--YF++++WF[+ZF++++XF]--XF";
        } else if ((atual == "+") || (atual == "-") ||  (atual == "[") ||
                   (atual == "]")) {
            proximaSentenca = proximaSentenca + atual;
        }
    }
    sentenca = proximaSentenca;
    ggbApplet.evalCommand("n=n+1");
    for (var contSent = 0; contSent < sentenca.length; contSent++) {
        movimentacao(contSent);
    }
}

function movimentacao(cont) {
    ggbApplet.evalCommand("StartAnimation[ ]");
    var atual = sentenca.charAt(cont);
    var correcaoAngulo, contAngulo, anguloArmazenado;
    if (atual == "F") {
        ggbApplet.evalCommand("TurtleForward[T, length]");
    } else if (atual == "-") {
        ggbApplet.evalCommand("TurtleLeft[T, angle]");
        anguloAtual = anguloAtual - 1;
    } else if (atual == "+") {
        ggbApplet.evalCommand("TurtleRight[T, angle]");
        anguloAtual = anguloAtual + 1;
    } else if (atual == "[") {
        x = ggbApplet.getXcoord("T");
        y = ggbApplet.getYcoord("T");
        pilhaX.push(x);
        pilhaY.push(y);
        pilhaAngulos.push(anguloAtual);
    } else if (atual == "]") {
        x = pilhaX.pop();
        y = pilhaY.pop();
        anguloArmazenado = pilhaAngulos.pop();
```

```
        ggbApplet.setCoords("T", x, y);
        correcaoAngulo = anguloArmazenado - anguloAtual;
        if (correcaoAngulo < 0) {
            correcaoAngulo = correcaoAngulo * (-1);
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleLeft[T, angle]");
                contAngulo = contAngulo + 1;
            }
        } else if (correcaoAngulo > 0) {
            contAngulo = 0;
            while (contAngulo < correcaoAngulo) {
                ggbApplet.evalCommand("TurtleRight[T, angle]");
                contAngulo = contAngulo + 1;
            }
        }
        anguloAtual = anguloArmazenado;
    }
}
```

Percebe-se algumas mudanças com relação a estrutura do Código 2 em comparação ao Código 1. Desta vez, é criada uma função denominada $sentencaMovimentacao()$, que desempenha dois papéis: o primeiro é atualizar a variável $sentenca$ substituindo nela o produto de todas as substituições provenientes das regras de produção, realizadas a cada geração. O segundo papel é chamar a função $movimentacao(cont)$, a qual opera da mesma forma que a função $movimentacao(sentenca, cont)$ do Código 1. A única exceção é que no Código 1 são inseridos dois parâmetros $sentenca$ e $contSent$ e no Código 2 é inserido apenas o parâmetro $contSent$. A função $sentencaMovimentacao()$ somente é realizada após clicar no botão "Gerar".

Isto se dá pelo fato que no Código 1 precisa-se primeiro passar o conteúdo que tinha armazenado em $sentenca$ para a função $movimentacao(sentenca, cont)$. Esta passagem tem como objetivo que a tartaruga pudesse realizar seus movimentos e as correções dos ângulos de seus giros, para depois, atualizá-la segundo as regras de produção do L-sistema Em outras palavras, a variável $sentenca$ precisava ser carregada em cada geração. No código 2, não há essa necessidade pois, em virtude de sua nova estrutura, primeiro é atualizada a variável $sentenca$ segundo as regras de produção do L-sistema e, após isso, são realizadas as movimentações e as correções dos ângulos de giro da tartaruga, utilizando a função $movimentacao(cont)$.

Outra mudança que foi crucial para o funcionamento correto do código desenvolvido no que diz respeito à correção dos ângulos, é a inserção da linha de comando: $anguloAtual = anguloArmazenado$, a qual atualiza a variável $anguloAtual$ com o novo valor de ângulo que está contido na variável

$anguloArmazenado$.

As simulações das primeira, segunda, terceira e quinta gerações da tesselação de Penrose obtidas a partir do Código 2 podem ser visualizadas nas Figuras 6.7, 6.8, 6.9 e 6.10 respectivamente.

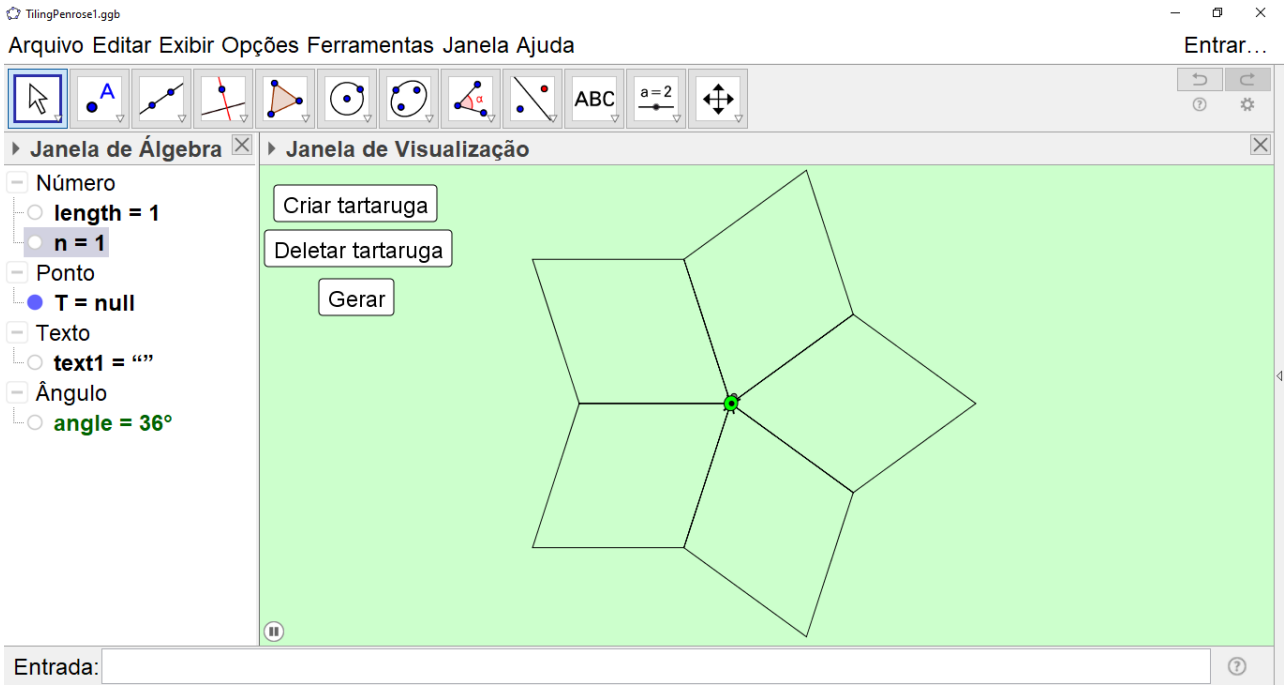


Figura 6.7: Modelagem da tesselação de Penrose da primeira geração ou axioma ($n = 1$ na "Janela de Álgebra").

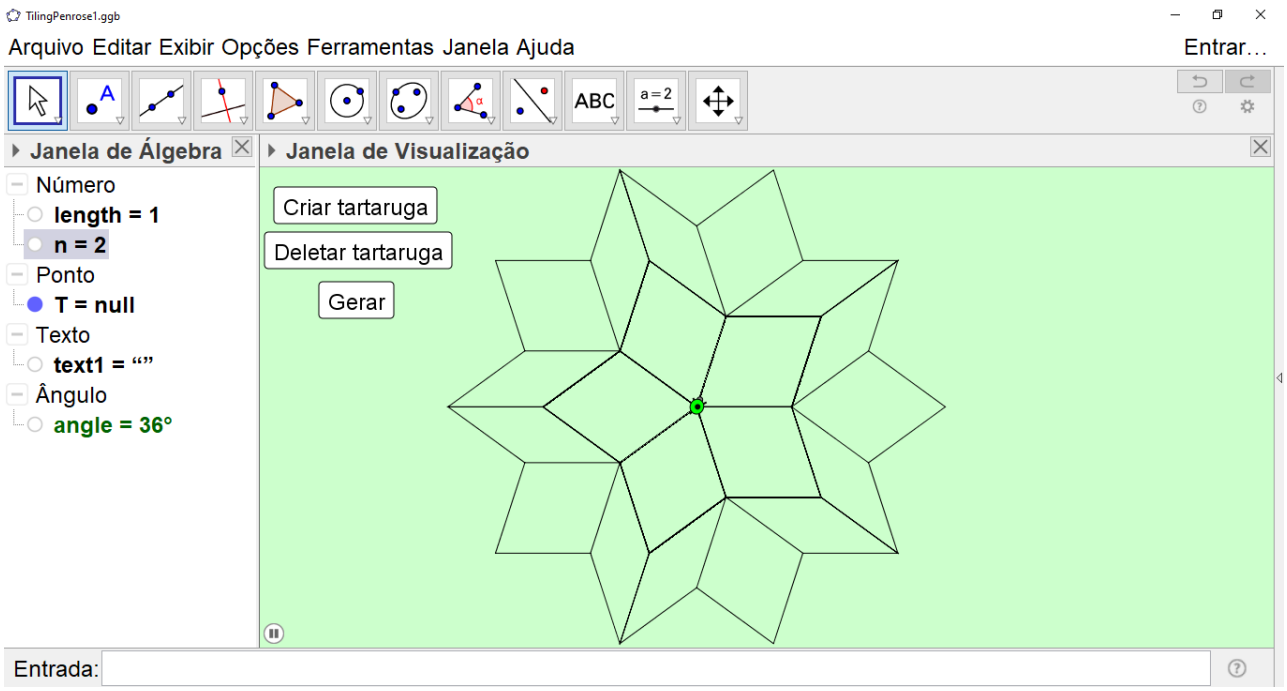


Figura 6.8: Modelagem da tesselação de Penrose da segunda geração ($n = 2$ na "Janela de Álgebra").

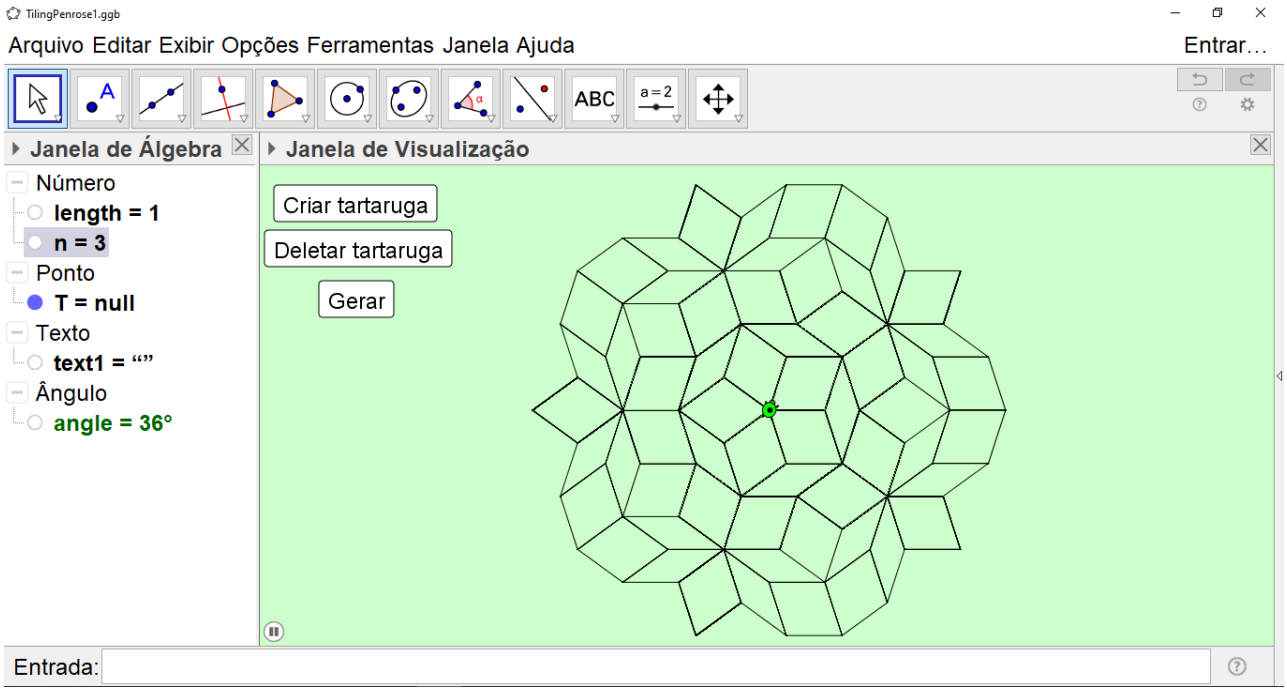


Figura 6.9: Modelagem da tesselação de Penrose da terceira geração ($n = 3$ na "Janela de Álgebra").

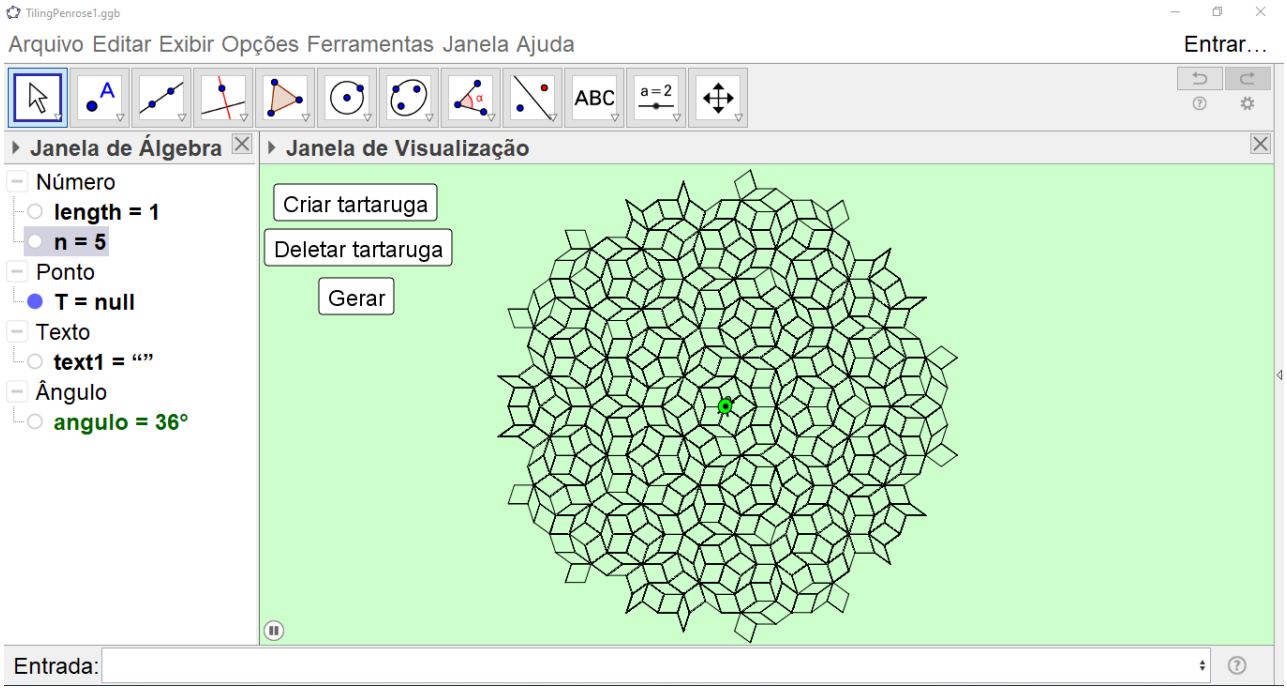


Figura 6.10: Modelagem da tesselação de Penrose da quinta geração ($n = 5$ na "Janela de Álgebra").

Com esta modelagem desafiadora e os problemas técnicos apresentados finaliza-se o caminho percorrido desde a tesselação unicelular (Capítulo 1) até este estágio avançado de programação aliado à construção das dinâmicas dos L-sistemas.

Conclusão

As inquietações referentes à busca incessante por conhecimento e experiência na área da matemática aplicada e computacional aguçaram orientando e orientadora a desenvolver o conteúdo abordado nesta pesquisa. Com o auxílio da tecnologia, que cresce diariamente a uma velocidade exponencial, é possível visualizar de maneira descontraída e divertida, objetos matemáticos que, há alguns anos, nem se cogitavam esta ocorrência.

Desta maneira, os objetivos propostos neste presente trabalho, como a modelagem matemática e computacional de fractais, curvas FASS (pode-se dizer curvas de preenchimento espacial), L-árvores e tesselações, mais especificamente, a tesselação de Penrose, atrelados a um sistema dinâmico discreto estabelecido por Lindenmayer, o L-sistema, convergiram, de modo uniforme, para o êxito. Indubitavelmente, o GeoGebra, em sincronia com suas ferramentas pouco exploradas, a tartaruga e o *JavaScript*, propiciam todo o alicerce tecnológico utilizado para que os objetivos tenham sido alcançados.

No que tangencia os aprendizados e crescimentos pessoais, encontram-se o processo de escrever, organizar e submeter trabalhos científicos para congressos (locais e nacionais). Além da experiência adquirida em buscar informação e realizar intensas pesquisas conquistando assim, bagagem literária e matemática para trabalhos atuais, futuros e para a vida. Em suma, todo este processo de desenvolvimento, propiciou contribuições para literatura em dois resumos estendidos (XX Semana da Matemática e X Semana da Estatística e IX Mostra IC; XI Semana de Matemática do Pontal) e um artigo completo (X ERMAC-RS). Além do mais, em virtude da atual pandemia de COVID-19 provocada pelo vírus Sars-CoV-2, houve a necessidade de extrema adequação para que fosse possível a exposição desses trabalhos nos congressos supracitados. Assim, possibilitando adquirir novas experiências na busca por meios tecnológicos para superar os obstáculos gerados pela crise sanitária.

Além de tudo isso, tem sido possível, no decorrer do processo de criação desta pesquisa, a percepção clara de que o envolvimento da tecnologia é extremamente essencial na vida de um estudante da Geração Alfa, cuja vivência é, substancialmente, rodeada pela mídia social.

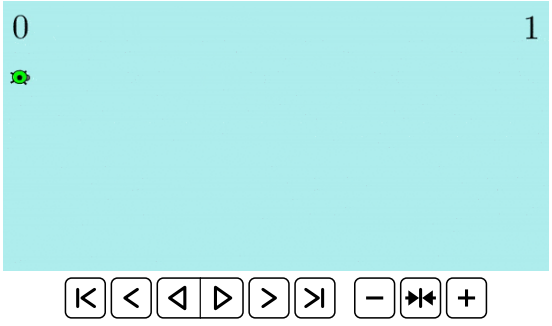
Apêndice

A fim de estimular a interação do leitor com o GeoGebra em conjunto com a tartaruga e o *JavaScript*, apresenta-se algumas simulações interativas das curvas modeladas ao longo deste trabalho.

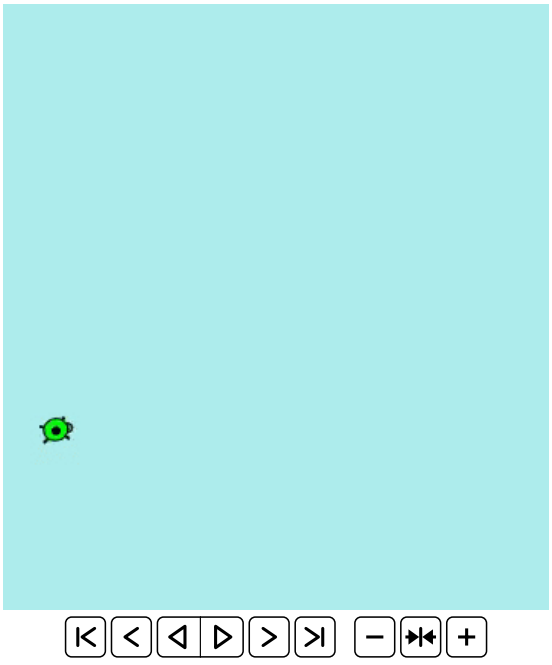
Para que isso fosse possível, foram utilizados dois *softwares*: *ImageMagick* que junto ao "Prompt de Comando" do Sistema Operacional *Windows*, cria, a partir de um GIF, centenas de *frames* para que fosse possível a utilização do comando `\animategraphics[]` proveniente do pacote `\usepackage{animate}` do LaTeX. Para obter o GIF foi necessário gravar a animação da modelagem para um vídeo através de uma captura de tela e, através de um outro *software* se faz a conversão de vídeo para GIF. Neste trabalho, utilizou-se um, deste tipo, disponível *online*.

Percebe-se que cada simulação possui a mesma sequência de botões, os quais promovem a relação máquina-usuário e significam: ⏮ volte para o primeiro quadro, ◀ volte um quadro para trás, ◁ caminhe para trás, ▷ caminhe para frente, ⏸ pause a animação, ▶ vá um quadro a frente, ⏭ vá para o último quadro, − diminua a velocidade da animação, ⏯ velocidade normal, + aumente a velocidade da animação.

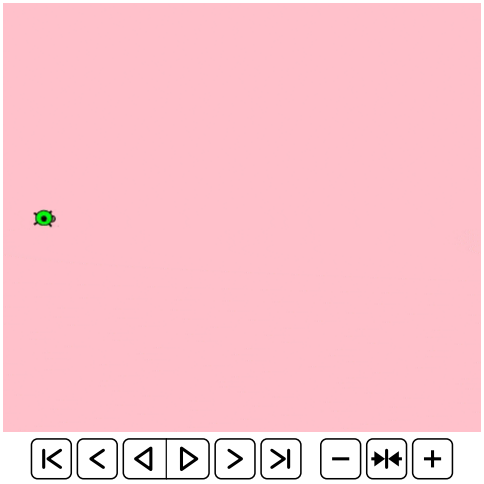
Conjunto de Cantor



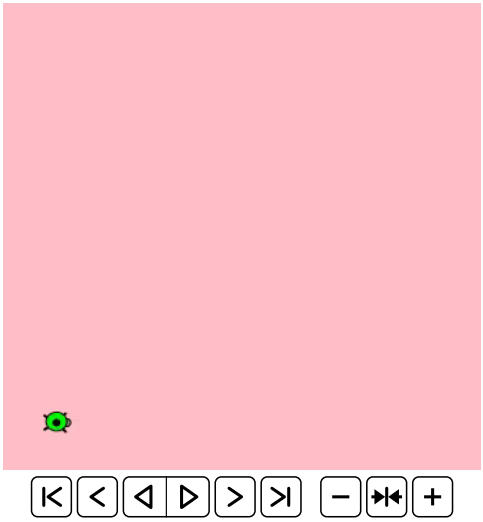
Floco de Neve de Koch



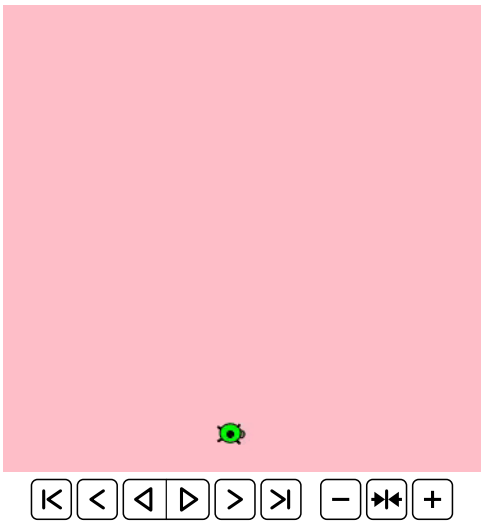
Curva de Peano



Curva de Hilbert



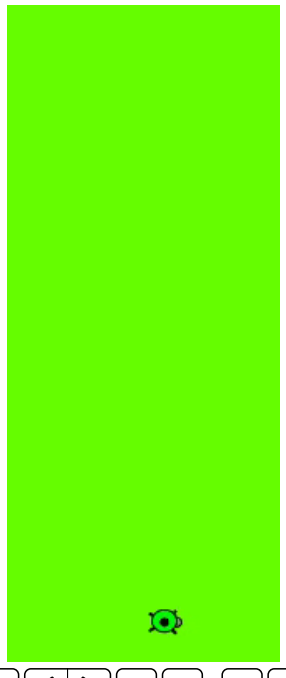
Curva de Moore



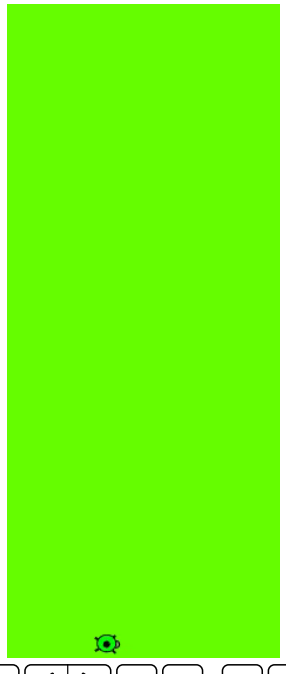
Curva do Dragão



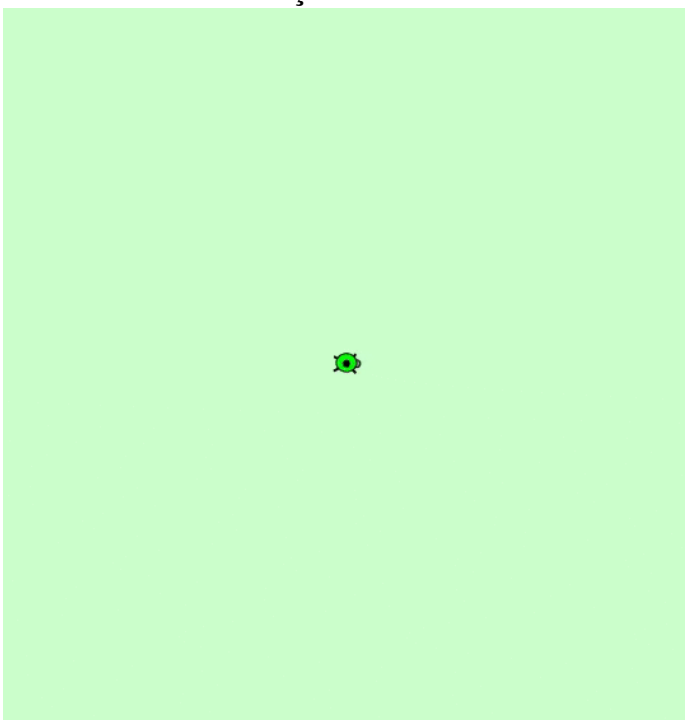
1ª L-árvore Axial



2ª L-árvore Axial



Tesselação de Penrose



Referências Bibliográficas

[1] AGRANOV, G. e GOTSMAN, C.: *Algorithms for rendering realistic terrain image sequences and their parellel implementation*. The Visual Computer, 1995.

[2] ASANO, T. et al.: *Space filling curves and their use in the design of geometric data structures*. Theoretical Computer Science, 1997.

[3] BADER, M.: *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer–Verlag, Berlin, Heidelberg, 2013.

[4] BERTILSSON, M., BRICKELL, E. e INGEMARSSON, I.: *Cryptanalysis of Video Encryption Based on Space-Filling Curves*. Em *Advances in Cryptology–EUROCRYPT '89*, pp. 403–411, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[5] BÖHM, C.: *Space-filling Curves for High-performance Data Mining*. Machine Learning arXiv:2008.01684, 2020.

[6] BIALLY, T.: *Space-filling curves: Their generation and their application to bandwidth reduction*. IEEE Transactions on Information Theory, pp. 658–664, 1969.

[7] BUTZ, A. R.: *Space filling curves and mathematical programmming*. Information and Control, pp. 314–330, 1968.

[8] CANTOR, G.: *Ueber unendliche, lineare Punktmannichfaltigkeiten*. Mathematische Annalen, 20(1):113–121, 1882.

[9] CHANG, A. e ZHANG, T.: *The Fractal Geometry of the Boundary of Dragon Curves*. Journal of Recreational Mathematics, 1999–2000.

[10] COWELL, J. R.: *Syntactic pattern recognizer for vehicle identification numbers*. Image and Vision Computing, pp. 13–19, 1995.

[11] CRILLY, A. J., EARNSHAW, R. A. e JONES, H.: *Fractals and Chaos*. Springer–Verlag, New York, 1991.

[12] DICKAU, R.: *Hilbert and Moore Fractal Curves*, 2011. *Website* acessado em 10/2020 no endereço: https://demonstrations.wolfram.com/HilbertAndMooreFractalCurves/.

[13] DOLCE, O. e POMPEU, J. N.: *Fundamentos de Matemática Elementar*, vol. 9. Atual, São Paulo, 1977.

[14] FRACCHIA, F. D., PRUSINKIEWICZ, P. e BOER, M. J. M.: *Animation of the Development of Multicellular Structures*. Em *Computer Animation '90*, pp. 3–19, Tokyo, 1990. Springer Japan.

[15] GOODMAN, D. et al.: *JavaScript Bible*. Wiley Publishing, Inc., Indianapolis–Indiana, 7^a ed., 2010.

[16] GRÜNBAUM, B. e SHEPHARD, G. C.: *Tilings and Patterns*. W. H. Freeman, New York, 1987.

[17] GUAN, X., OOSTEROM, P. e CHENG, B.: *A Parallel N-Dimensional Space-Filling Curve Library and Its Application in Massive Point Cloud Management*. ISPRS International Journal of Geo-Information, p. 327, ago. 2018.

[18] GUGGENHEIMER, H. W.: *Differential Geometry*. Dover Publications, New York, 1977.

[19] HAMILTON, P.: *A language to describe the growth of neurites*. Biological Cybernetic, pp. 559–565, 1993.

[20] HILBERT, D.: *Ueber die stetige Abbildung einer Linie auf ein Flächenstück*. Mathematische Annalen, 38:459–460, 1891.

[21] HOHENWARTER, M.: *Dynamic mathematics with GeoGebra*. Journal of Online Mathematics and its Applications, 2007.

[22] HOHENWARTER, M.: *GeoGebra Classic 5.0*. http://www.geogebra.org, 2020. Acessado em 10/2020.

[23] HOLLIDAY, D. J. e SAMAL, A.: *Object recognition using L-system fractals*. Pattern Recognition Letters, 16:33–42, 1995.

[24] HORNBY, G. e POLLACK, J.: *Evolving L-systems to generate virtual creatures*. Computers & Graphics, 25:1041–1048, 2001.

[25] KALAY, A., PARNAS, H. e SHAMIR, E.: *Neuronal growth via hybrid system of self-growing and diffusion based grammar rules: I.* Bulletin of Mathematical Biology, 57:205–227, 1995.

[26] KEPLER, J.: *Harmonices Mundi*. Tambach, Gottfried, fl. 1607-1632, 1619.

[27] KOCH, H.: *Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes*. Acta Mathematica, pp. 145–174, 1906.

[28] KRILL, P.: *JavaScript creator ponders past, future*, 2008. *Website* acessado em 11/2020 no endereço: https://www.infoworld.com/article/2653798/javascript-creator-ponders-past–future.html.

[29] LIMA, E. L.: *Espaços Métricos*. Instituto de Matemática Pura e Aplicada - Projeto Euclides, Rio de Janeiro–Brasil, 2^a ed., 1983.

[30] LINDENMAYER, A.: *Adding Continuous Components to L-Systems*. Em *L Systems, Most of the papers were presented at a conference in Aarhus, Denmark, January 14-25, 1974*, vol. 15 de *Lecture Notes in Computer Science*, pp. 53–68. Springer, 1974.

[31] LU, P. e STEINHARDT, P.: *Decagonal and Quasi-Crystalline Tilings in Medieval Islamic Architecture*. Science (New York, N.Y.), 315:1106–1110, 2007.

[32] MANDELBROT, B.: *The Fractal Geometry of Nature*. San Francisco: W.H. Freeman, California, 1982.

[33] MANOUSAKIS, S.: *Musical L-Systems*. Tese de Mestrado, The Royal Conservatory, Haia–Países Baixos, jun. 2006.

[34] MCCORMICK, B. e MULCHANDANI, K.: *L-system modeling of neurons*. Em *Visualization in Biomedical Computing 1994*, vol. 2359, pp. 693–705. SPIE, 1994.

[35] MOORE, E. H.: *On Certain Crinkly Curves*. Transactions of the American Mathematical Society, 1:72–90, 1900.

[36] PAPERT, S.: *Logo: Computadores e educação*. Editora Brasiliense, São Paulo–Brasil, 1985. Publicado originalmente sob o título de Mindstorms: children, computers and powerful ideas. New York: Basic Books, 1980.

[37] PEANO, G.: *Arithmetices principia: nova methodo exposita*. Fratres Bocca, 1889.

[38] PEANO, G.: *Sur une courbe, qui remplit toute une aire plane*. Math. Ann., pp. 157–160, 1890.

[39] PENROSE, R.: *Roger Penrose: Collected Works: Volume 4: 1981-1989*. Oxford University Press, 2010.

[40] PENROSE, R.: *Roger Penrose: Collected Works: Volume 5: 1990-1996*. Oxford University Press, 2010.

[41] PENROSE, R.: *Roger Penrose: Collected Works: Volume 6: 1997–2003*. Oxford University Press, 2010.

[42] PENROSE, R.: *The road to reality: a complete guide to the laws of the universe*. Vintage Books USA, Illustrator edition: 2007.

[43] PRUSINKIEWCZ, P. e SANDNESS, G.: *Koch curves as attractors and repellers*. IEEE Computer Graphics and Applications, 1988.

[44] PRUSINKIEWICZ, P. e LINDENMAYER, A.: *The Algorithmic Beauty of Plants*. Sringer–Verlag, New York, 1990.

[45] SAGAN, H.: *Space-Filling Curves*. Springer–Verlag, New York, 1994.

[46] SENECHAL, M.: *Quasicrystals and geometry*. Cambridge University Press, 1996.

[47] SMITH, H. J. S.: *On the Integration of Discontinuous Functions*. Proceedings of the London Mathematical Society, pp. 140–153, 1874.

[48] TABACHNIKOV, S.: *Dragon Curves Revisited*. Mathematical Intelligencer, 2014.

[49] REDE DE DESENVOLVEDORES DA MOZILLA (MDN): *O que é JavaScript?* https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/O_que_e_JavaScript, 2020. Acessado em: 11/2020.

[50] ZEE, A.: *Fearful Symmetry: The Search for Beauty in Modern Physics*. Princeton University Press, New Jersey, abr. 2016.

Índice Remissivo

Aristid Lindenmayer, 33

Benoît Mandelbrot, 36

Curvas de Preenchimento, 3, 46, 55, 89

David Hilbert, 46

Eliakim Moore, 46

Fractais, 3, 36, 40, 46, 60, 89

GeoGebra, 3, 5–8, 12, 17, 20, 22, 27, 28, 31, 32, 35, 38, 43–45, 47, 50, 52, 55, 59, 66, 69, 72, 77, 82, 83, 89

Georg Cantor, 37

Giuseppe Peano, 46

Helge von Koch, 40

JavaScript, 5, 26, 27, 30–32, 36–38, 40, 61, 65, 67, 70, 74, 77, 89

L-sistemas, 3, 4, 33–37, 40, 43, 46, 47, 49, 50, 52, 55, 60, 61, 67, 70, 73, 74, 79, 86, 89

L-árvore, 60, 61, 69, 89

Manual da Tartaruga, 7

Markus Hohenwarter, 5

Roger Penrose, 78

Seymour Papert, 6

Tartaruga do GeoGebra, 3, 5–12, 14, 15, 17–25, 27–30, 32, 34, 36–40, 42, 43, 49, 52, 54, 58, 64–66, 74, 77, 81, 86, 89

Tesselação, 3, 15–17, 20, 22, 27, 29–31, 73, 74, 77, 78, 89

Tesselação de Penrose, 74, 78, 79, 82, 83, 87–89

Turtle Graphics, 3, 6, 7, 47