

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SIMULADOR DE AUTÔMATOS CELULARES
UNIDIMENSIONAIS

GUILHERME HUMBERTO JANSEN

BLUMENAU
2016

GUILHERME HUMBERTO JANSEN

SIMULADOR DE AUTÔMATOS CELULARES

UNIDIMENSIONAIS

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Profa. Joyce Martins, Mestre – Orientadora

Prof, Paulo César Rodacki Gomes, Doutor - Coorientador

**BLUMENAU
2016**

SIMULADOR DE AUTÔMATOS CELULARES

UNIDIMENSIONAIS

Por

GUILHERME HUMBERTO JANSEN

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente: _____
Prof. Joyce Martins, Mestre – Orientadora, FURB

Membro: _____
Prof. Alexander Roberto Valdameri, Mestre – FURB

Membro: _____
Prof. Andreza Sartori, Doutora – FURB

Blumenau, 9 de dezembro de 2016

Dedico este trabalho ao meu pai e à minha
mãe (em memória).

AGRADECIMENTOS

A Deus, por todos os aspectos das quais sou capaz de contemplar de forma subjetiva ou objetiva como favoráveis à harmonia e, principalmente, pelos aspectos das quais não sou capaz de contemplar.

À minha família, por todo o apoio, incentivo, paciência, ajuda, compreensão, respeito, educação e outras tantas características que contribuíram para este trabalho, assim como contribuem para a minha vida.

Aos meus colegas de trabalho, por estimularem minha inspiração pessoal e profissional diariamente e por todas as oportunidades respeitosamente concedidas e confiadas a mim, com grande ênfase ao tempo concedido para construir este projeto e à paciência durante esta fase.

Aos meus amigos, por aceitarem compartilhar situações boas e ruins de forma nobre, pelos devaneios embriagados, aventuras de finais de semana, confiança e por me ajudarem a chegar a conclusões importantes.

À minha orientadora, que gentilmente conduziu a evolução deste projeto de forma eficaz, comprometida, clara, compreensiva e da qual tive a honra de argumentar sobre as mais diversas características deste projeto, recebendo sempre sua valiosa ajuda.

Ao meu coorientador, Paulo César Rodacki Gomes, por aceitar este desafio e compartilhar sua experiência através da avaliação de problemas misteriosos, testes de soluções alternativas, discussões sobre performance e com toda a ajuda ao longo do projeto.

À Lucila, pela desiderata.

A única razão para o tempo é para que não aconteça tudo no mesmo instante.

Albert Einstein

RESUMO

Este trabalho descreve a implementação de um simulador de autômatos celulares unidimensionais, desenvolvido sob uma perspectiva abstrata multidimensional. O objetivo do projeto é simular os autômatos celulares elementares definidos por Wolfram (2002), permitindo que o usuário informe os parâmetros do autômato celular elementar a ser simulado e possa usar recursos gráficos de propósito didático para entender os conceitos envolvidos. A fundamentação teórica foi baseada nos conceitos de Wolfram (2002) assim como de outros autores importantes, reunindo aqueles necessários para a elicitação de requisitos e para a especificação do simulador. O projeto foi desenvolvido sob a licença de software livre AGPLv3, sendo estruturado em cinco módulos Maven implementados na linguagem Java usando a biblioteca gráfica Processing. Foram efetuadas simulações de três regras elementares, geradas pela ferramenta correlata Mathematica (WOLFRAM ALPHA LLC, 2016) e comparadas com as simulações efetuadas na ferramenta desenvolvida. As comparações foram efetuadas sob aspecto lógico através da biblioteca JUnit, sob aspecto visual utilizando a ferramenta BeyondCompare e sob aspecto de tempo através de uma média entre os tempos de execução. Os resultados dos testes mostraram que o simulador desenvolvido funciona de maneira adequada, embora seja mais lento do que o trabalho correlato. O simulador desenvolvido possui uma interface gráfica de fácil utilização com recursos de propósito didático, que contribuem para o estudo e disseminação da área de autômatos celulares.

Palavras-chave: Autômatos celulares. Simuladores. Complexidade computacional

ABSTRACT

This paper describes the implementation of a simulator of one-dimensional cellular automata, developed from a multidimensional abstract perspective. The objective of the project is to simulate the elementary cellular automata defined by Wolfram (2002), allowing the user to inform the parameters of the elementary cellular automata to be simulated, and also to interact with graphical resources of didactic purpose for a better understanding of the concepts. The theoretical basis gather concepts from Wolfram (2002) and also from other important authors, needed for the requirements elicitation and for the specification of the simulator. The project was developed under the free software license AGPLv3, being structured in five Maven modules implemented in Java language and using the Processing library. Simulations of 3 elementary rules were generated by the correlative work Mathematica (WOLFRAM ALPHA LLC, 2016) and compared with the simulation performed in the developed tool. The comparisons were made under logical aspect through the JUnit library, under visual aspect using the tool BeyondCompare and also under aspect of time through an average of runtimes. The test results showed that the developed simulator works normally, although it is slower than the correlated work. The developed simulator has an easy-to-use graphical interface with didactic features that contribute to the study and dissemination of cellular automata area.

Key-words: Cellular automata. Simulators. Computational complexity.

LISTA DE FIGURAS

Figura 1 – Espaços com número de dimensões diferentes	18
Figura 2 – Variedade geométrica celular e seus espaços.....	18
Figura 3 – Ação do tempo nos espaços unidimensional e bidimensional	19
Figura 4 – Espaço polar	19
Figura 5 – Espaço toroidal.....	19
Figura 6 – Estrutura de uma regra em espaços com dimensões diferentes	20
Figura 7 – Vizinhanças	21
Figura 8 – Iterações da regra elementar 250.....	22
Figura 9 – Representação visual da regra elementar 250	22
Figura 10 – Ação da regra elementar 250 entre iterações	22
Figura 11 – Regras elementares 250 (à esquerda) e 110 (à direita)	23
Figura 12 – Condição de borda periódica.....	24
Figura 13 – Condição de borda reflexiva	24
Figura 14 – Condição de borda fixa	24
Figura 15 – Regra elementar de base binária	25
Figura 16 – Regra 110	27
Figura 17 – Padrões de pigmentação em conchas de moluscos	27
Figura 18 – Flocos de neve de formas e tamanhos diferentes	28
Figura 19 – Representação de um floco de neve através de um autômato celular	28
Figura 20 – Espaço celular de um simulador de fluidos.....	29
Figura 21 – Simulação de turbulência em fluidos	30
Figura 22 – Processo de desenvolvimento de simulações.....	31
Figura 23 – Processo de verificação, validação e qualificação	32
Figura 24 – Golly.....	33
Figura 25 – Simulação da regra 110 no sistema Golly.....	34
Figura 26 – Mathematica.....	35
Figura 27 – Computação da regra 110 na ferramenta Mathematica.....	36
Figura 28 – Diagrama de dependências entre os módulos do projeto	38
Figura 29 – Diagrama de conjuntos das classes do módulo <code>cas-core</code>	39
Figura 30 – Representação de classes do modelo de dados através da regra 110.....	39
Figura 31 – Diagrama de classes: <code>Simulation</code> , <code>Universe</code> e <code>CellularAutomaton</code> ..	40

Figura 32 – Diagrama de classes: Universe, Space e Time	40
Figura 33 – Mecanismo incremental de tempo unidimensional (A) e bidimensional (B)	42
Figura 34 – Representação do tempo sobre o espaço	42
Figura 35 – Fórmula da execução incremental máxima de tempo	43
Figura 36 – Representação dos atributos da classe Space através da regra 110	43
Figura 37 – Diagrama de classes: CellularAutomaton, Rule, Transition, Combination e State	45
Figura 38 – Algoritmo da classe CellularAutomaton	46
Figura 39 – Fórmula para execução da regra do autômato celular	46
Figura 40 – Algoritmo de construção do modelo de dados	47
Figura 41 – Diagrama de casos de uso	49
Figura 42 – Diagrama de atividades	49
Figura 43 – Diagrama de estados	50
Figura 44 – Interface do simulador	52
Figura 45 – Simulação da regra 110 sem zoom	54
Figura 46 - Simulação da regra 110 com zoom e inspeção	55
Figura 47 – Condição inicial no arquivo CAS	56
Figura 48 – Validação lógica das regras elementares 30, 90 e 110	57
Figura 49 – Validação visual da regra elementar 30	58
Figura 50 – Validação visual da regra elementar 90	59
Figura 51 – Validação visual da regra elementar 110	59
Figura 52 – Representação gráfica das 256 regras elementares	66
Figura 53 – Padrões de pigmentação encontrados em animais	70
Figura 54 – Padrões em estruturas animais e vegetais	71
Figura 55 – Padrões e turbulência em fluidos	72

LISTA DE QUADROS

Quadro 1 – Descrição dos módulos do projeto.....	38
Quadro 2 – Métodos abstratos da classe <code>Space</code>	44
Quadro 3 – Comparativo entre os trabalhos correlatos e o simulador CAS.....	61

LISTA DE TABELAS

Tabela 1 – Consumo de tempo para simulação da regra 110	60
---	----

LISTA DE ABREVIATURAS E SIGLAS

RF – Requisito Funcional

RNF – Requisito Não Funcional

SUMÁRIO

1 INTRODUÇÃO.....	15
1.1 OBJETIVOS.....	16
1.2 ESTRUTURA.....	16
2 FUNDAMENTAÇÃO TEÓRICA.....	17
2.1 AUTÔMATOS CELULARES.....	17
2.1.1 Espaço.....	17
2.1.2 Célula.....	18
2.1.3 Tempo.....	18
2.1.4 Regra.....	19
2.1.5 Vizinhança.....	20
2.1.6 Mecânica estatística dos autômatos celulares unidimensionais.....	21
2.1.7 Condições de borda.....	23
2.1.8 Cálculo de regras para autômatos celulares unidimensionais.....	24
2.1.9 Aplicações para autômatos celulares.....	25
2.2 SIMULAÇÃO.....	30
2.3 TRABALHOS CORRELATOS.....	32
2.3.1 Golly.....	32
2.3.2 Mathematica.....	34
3 DESENVOLVIMENTO.....	37
3.1 REQUISITOS.....	37
3.2 ESPECIFICAÇÃO.....	38
3.2.1 Estrutura do projeto.....	38
3.2.2 Módulo cas-core.....	38
3.2.3 Módulo cas-control.....	46
3.2.4 Módulo cas-unidimensional.....	48
3.2.5 Módulo cas-ui-desktop.....	48
3.3 IMPLEMENTAÇÃO.....	50
3.3.1 Técnicas e ferramentas utilizadas.....	50
3.3.2 Operacionalidade da implementação.....	51
3.4 ANÁLISE DOS RESULTADOS.....	56
3.4.1 Avaliação lógica.....	56

3.4.2 Avaliação visual	57
3.4.3 Avaliação de tempo	59
3.4.4 Comparativo entre os trabalhos correlatos	61
4 CONCLUSÕES	63
4.1 EXTENSÕES	63
REFERÊNCIAS.....	65
APÊNDICE A – REGRAS ELEMENTARES	66
APÊNDICE B – PADRÕES BIOLÓGICOS.....	70
APÊNDICE C – PADRÕES E TURBULÊNCIA EM FLUIDOS.....	72

1 INTRODUÇÃO

Há três séculos, a ciência foi transformada pela ideia de que regras baseadas em equações matemáticas poderiam ser utilizadas para descrever o mundo natural (WOLFRAM, 2002). Foi no início dos anos 50, conforme descreve Pickover (2009), no Laboratório Nacional de Los Alamos, que Stanislaw Ulam e John Von Neumann criaram o conceito de autômato celular, o primeiro durante o estudo do crescimento de cristais e o segundo de sistemas auto reprodutores. Os detalhes sobre os estudos de John Von Neumann permaneceram inéditos até a sua morte em 1957, sendo editados e publicados posteriormente em 1966 por Arthur W. Burks (MCINTOSH, 2009).

Foi apenas em outubro de 1970 que uma publicação de Martin Gardner na revista *Scientific American* impulsionou a popularização do tema, descrevendo a criação de John Horton Conway intitulada *Game of Life*, o autômato celular bidimensional mais conhecido até hoje. Ainda assim, Schiff (2008) afirma que o estudo dos autômatos celulares não continha muita profundidade, análise e aplicabilidade, impossibilitando sua abordagem como disciplina científica. Schiff (2008) comenta ainda que esta realidade mudou no início dos anos 80, quando o físico Stephen Wolfram iniciou o primeiro estudo aprofundado sobre os autômatos celulares (WOLFRAM, 1983). Nesta obra, assim como em obras subsequentes, Wolfram criou algumas das imagens que se tornaram icônicas na área, representadas inclusive em seu livro (WOLFRAM, 2002).

Wolfram (2002) apresenta mais de vinte áreas do conhecimento diferentes na qual os autômatos celulares podem ser utilizados, entre elas: matemática, física, biologia e ciência da computação. Ele destaca ainda que este conceito possui capacidade de generalização superior à da matemática contemporânea, sendo capaz inclusive de abordar problemas em seus fundamentos. Na definição dos autômatos celulares, Wolfram (1983) os formaliza como idealizações matemáticas de sistemas físicos nas quais o espaço e o tempo são discretos, sendo as grandezas físicas abordadas por um conjunto finito de valores igualmente discretos. Sua representação pode ser feita por meio de um espaço n -dimensional composto por células que interagem entre si em função do tempo, alterando seus estados através de uma regra pré-estabelecida.

Diante da grande abrangência desta área em recente expansão, desenvolveu-se uma ferramenta que serve de apoio didático para iniciar o estudo dos autômatos celulares na sua forma mais elementar. O objetivo desta ferramenta é computar autômatos celulares unidimensionais, permitindo que o usuário entenda o seu funcionamento de forma prática,

garantindo a base de conhecimento necessária para explorar abordagens de maior complexidade.

1.1 OBJETIVOS

Este trabalho objetiva criar uma ferramenta para auxiliar o estudo dos autômatos celulares.

Os objetivos específicos do trabalho são:

- a) construir um simulador de autômatos celulares unidimensionais elementares;
- b) permitir que o usuário forneça os parâmetros do autômato celular a ser computado;
- c) criar recursos de propósito didático integrados ao simulador, tais como execução passo-a-passo, configuração da regra a ser utilizada e número de iterações a serem executadas.

1.2 ESTRUTURA

O conteúdo deste documento está dividido em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusão. Inicialmente é apresentada a introdução, seguida pela fundamentação teórica, que descreve os conceitos fundamentais sobre autômatos celulares utilizados nesse trabalho. Também são apresentadas ferramentas similares ao simulador desenvolvido. O capítulo seguinte traz a especificação e o desenvolvimento do simulador. Por fim, são discutidos os resultados e conclusões obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

Inicialmente, é apresentada a definição geral dos autômatos celulares, baseada principalmente nos conceitos criados por Stephen Wolfram. Em seguida, são elaboradas as principais características dos autômatos celulares, incluindo também, uma visão geral sobre simulação. Por fim, são abordados dois trabalhos correlatos a este.

2.1 AUTÔMATOS CELULARES

Segundo Wolfram (1983), os autômatos celulares são idealizações matemáticas de sistemas físicos nas quais o espaço e o tempo são discretos, sendo as grandezas físicas abordadas por um conjunto finito de valores igualmente discretos. Wolfram (1983) complementa que um autômato celular consiste em um espaço uniforme (ou grade), comumente infinito em extensão, com uma variável discreta em cada unidade de espaço (célula).

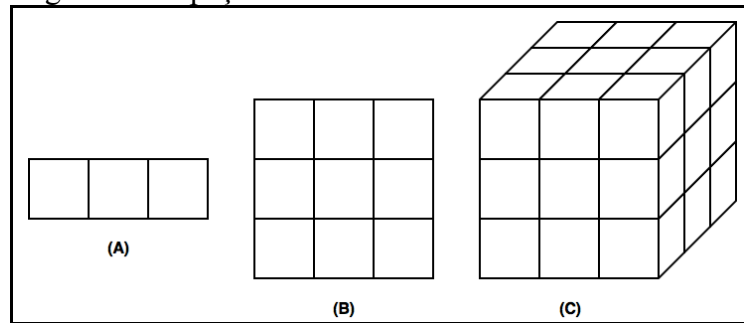
O estado de um autômato celular é completamente especificado pelos valores das variáveis das células. Um autômato celular evolui em intervalos de tempo discretos, com o valor da variável de uma célula sendo afetado pelos valores nas variáveis de células de sua vizinhança, na iteração de tempo anterior. A vizinhança de uma célula é tipicamente definida como sendo a própria célula e todas as células adjacentes imediatas. As variáveis de cada célula são atualizadas simultaneamente, baseadas nos valores das variáveis na vizinhança na iteração de tempo anterior e de acordo com um dado conjunto de regras locais.

2.1.1 Espaço

O espaço de um autômato celular, também citado como espaço celular (FLOREANO; MATTIUSI, 2008) ou grade (WOLFRAM, 1983), é um espaço n-dimensional composto por subdivisões espaciais chamadas células, que se distribuem uniformemente preenchendo suas dimensões. A Figura 1 exibe exemplos de espaços unidimensional (A), bidimensional (B) e tridimensional (C).

A disposição do espaço pode variar de acordo com a forma geométrica de suas células. Da mesma forma, algumas combinações entre a geometria de uma célula e a evolução do tempo no espaço permitem explorar o funcionamento de um autômato celular utilizando métodos alternativos.

Figura 1 – Espaços com número de dimensões diferentes



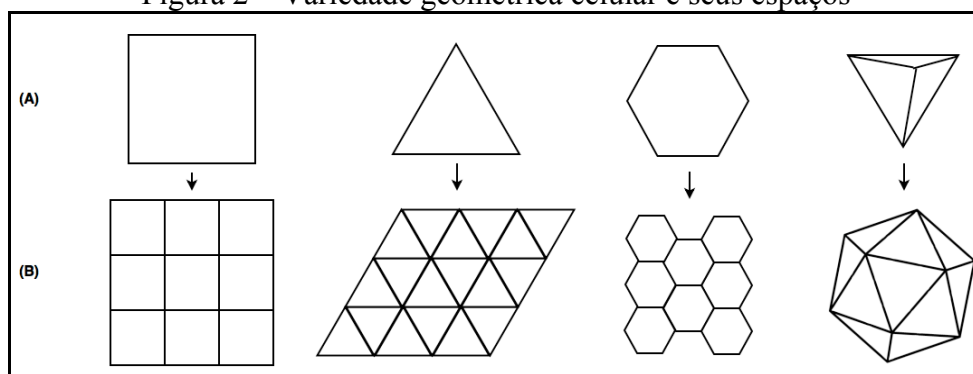
Fonte: adaptado de Floreano e Mattiussi (2008).

2.1.2 Célula

A célula é uma unidade espacial do espaço de um autômato celular, capaz de manifestar diferentes estados, comumente representados por cores, que podem variar ao longo do tempo pela ação de uma regra (WOLFRAM, 2002).

Na Figura 1, é possível notar que o espaço unidimensional representado por (A) possui 3 células quadradas, o espaço bidimensional representado por (B) possui 9 células quadradas e o espaço tridimensional representado por (C) possui 27 células cúbicas. Da mesma forma com que os quadrados e cubos foram utilizados como a forma geométrica vigente para todas as células de cada um de seus espaços, outras formas geométricas podem ser utilizadas para dar origem a espaços diferentes. A Figura 2 apresenta algumas formas geométricas que podem ser utilizadas como células de um espaço (A), demonstrando também a maneira com que ocorre o agrupamento em seus respectivos espaços (B).

Figura 2 – Variedade geométrica celular e seus espaços



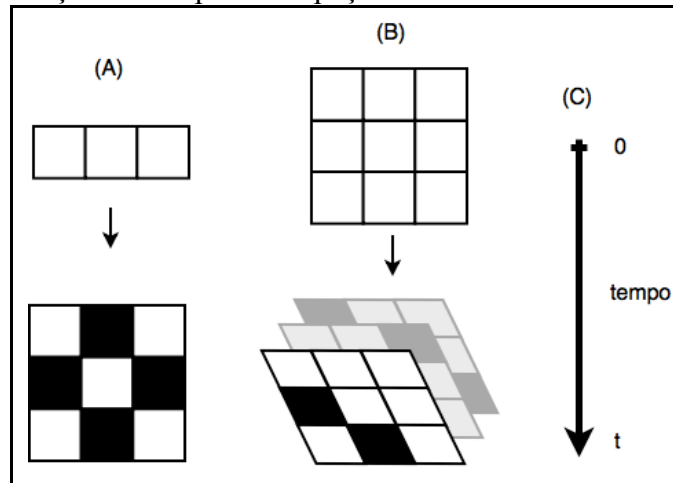
Fonte: adaptado de Floreano e Mattiussi (2008).

2.1.3 Tempo

O tempo, também citado por Wolfram (2002) como passo ou iteração, é um ciclo de processamento absoluto, considerando todas as células do espaço e todas as regras nele previstas. Conforme novas iterações de tempo acontecem no espaço, as células são avaliadas pelas regras podendo assumir novos estados.

A Figura 3 mostra uma representação do conceito de múltiplas iterações de tempo, de acordo com as dimensões do espaço. No espaço unidimensional (A), as iterações anteriores são mantidas visíveis formando um histórico de aparência bidimensional. Já nos espaços bidimensionais (B) e tridimensionais (não representados na figura), as representações anteriores do espaço são sobrescritas, tornando apenas a última iteração visível com a ação do tempo (C).

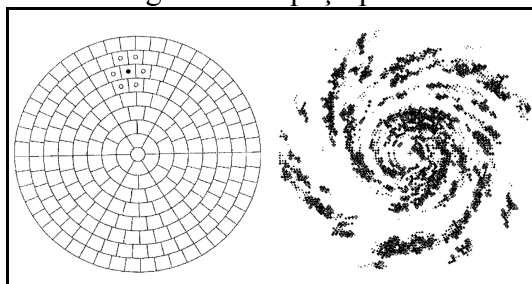
Figura 3 – Ação do tempo nos espaços unidimensional e bidimensional



Fonte: adaptado de Floreano e Mattiussi (2008).

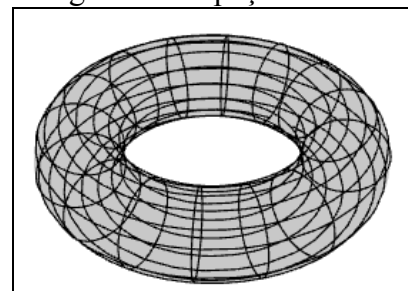
O formato da célula utilizada nos espaços unidimensionais, quando associada a múltiplas iterações no tempo, pode originar estruturas circulares. A Figura 4 mostra um espaço polar, utilizado no estudo da formação de galáxias, enquanto a Figura 5 apresenta um espaço em formato toroidal, gerado utilizando a ferramenta Mathematica.

Figura 4 – Espaço polar



Fonte: Schiff (2008, p. 42).

Figura 5 – Espaço toroidal



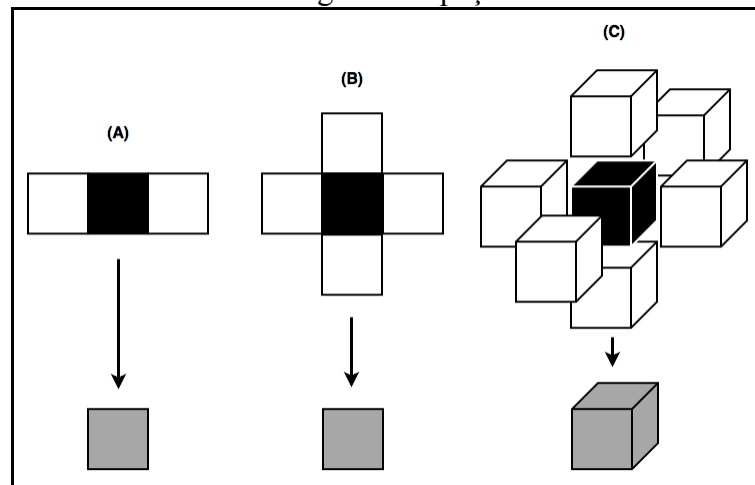
Fonte: elaborado pelo autor.

2.1.4 Regra

A regra de um autômato celular é o conjunto de combinações de estados possíveis entre uma célula de referência e suas vizinhas, estabelecendo um novo estado (cor) para a célula de referência na iteração de tempo seguinte àquela em que a combinação foi identificada (WOLFRAM, 1983).

A Figura 6 mostra o formato de uma regra do espaço unidimensional (A), bidimensional (B) e tridimensional (C). As células de cor preta representam uma célula de referência contida em seu respectivo espaço, as células na cor branca representam a vizinhança da célula de referência, que variam em quantidade e posição de acordo com as dimensões do espaço. Por fim, as células de cor cinza representam o novo estado (cor) da célula de referência que será utilizada na nova iteração de tempo deste espaço, quando a combinação de cores prevista pela regra for identificada.

Figura 6 – Estrutura de uma regra em espaços com dimensões diferentes



Fonte: adaptado de Floreano e Mattiussi (2008).

Considerando cada um dos espaços apresentados na Figura 6 sob a ação do tempo, suas respectivas regras serão utilizadas entre a iteração de tempo concluída e a nova iteração a ser computada, avaliando se existe alguma célula de cor preta que possui todos os seus vizinhos na cor branca. Caso esta combinação de cores seja localizada na iteração anterior, a célula de cor preta será apresentada com a cor cinza na próxima iteração.

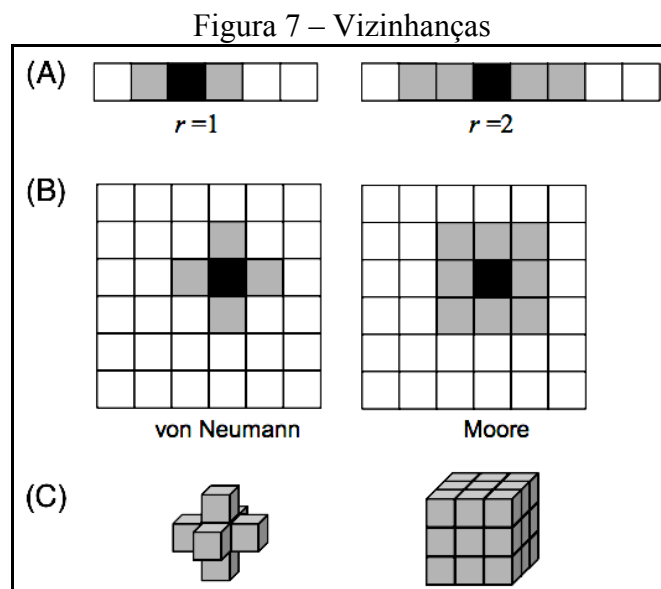
Wolfram (1983) criou tipos para categorizar as regras, sendo considerados na presente abordagem os tipos: elementar, geral e totalista. As regras elementares representam a forma mais simples de um autômato celular, utilizando apenas duas cores em um espaço unidimensional. As regras gerais possuem uma ou mais cores em espaços de uma ou mais dimensões. Da mesma forma, as regras totalistas também possuem uma ou mais cores em espaços de uma ou mais dimensões, porém, sem considerar o posicionamento das células vizinhas, mas sim a soma dos valores que representam a cor de cada célula da vizinhança.

2.1.5 Vizinhança

A vizinhança de uma célula é o conjunto de células, incluindo a própria célula, cujo estado pode influenciar diretamente o estado futuro da célula. Em outras palavras, cada célula

pode ser idealizada como diretamente conectada e sensível ao estado das células que pertencem à sua vizinhança (FLOREANO; MATTIUSI, 2008). Devido à variedade geométrica das células e do número de dimensões do espaço celular, algumas variações podem ser igualmente verificadas na vizinhança, sendo estas variações relacionadas à quantidade e ao posicionamento das células.

Conforme apresentado na Figura 7 (A), em um espaço unidimensional a quantidade de células presentes à direita e à esquerda da célula de referência pode ser representada pela letra r , que, segundo Floreano e Mattiussi (2008), representa o raio (ou alcance) da vizinhança. Para espaços bidimensionais (B) ou tridimensionais (C), os dois casos mais comuns são as vizinhanças von Neumann e Moore. A vizinhança von Neumann é formada pela célula de referência e as células perpendiculares a ela, já a vizinhança Moore considera todas as células da vizinhança von Neumann e ainda as células em posição diagonal à célula de referência.



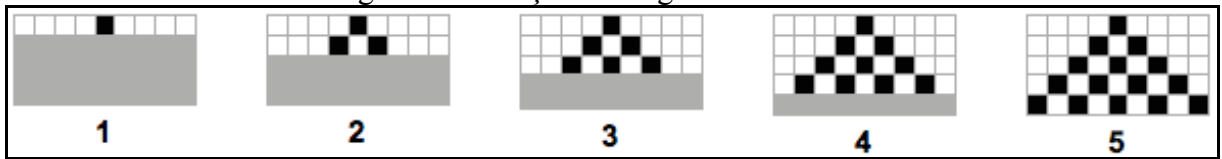
Fonte: adaptado de Floreano e Mattiussi (2008).

No simulador desenvolvido é considerada a vizinhança do espaço unidimensional de raio igual a 1, sendo esta vizinhança formada por uma célula à esquerda da célula de referência, outra à direita e pela própria célula de referência.

2.1.6 Mecânica estatística dos autômatos celulares unidimensionais

Os conceitos apresentados nas seções anteriores são utilizados para detalhar a execução dos autômatos celulares unidimensionais. Assumindo a execução da regra elementar 250 (WOLFRAM, 2002), é possível notar que as cores das células mudam com a evolução do tempo, considerando uma configuração inicial que contém apenas uma célula de cor preta. A Figura 8 mostra 5 iterações da regra elementar 250 a partir da configuração inicial descrita.

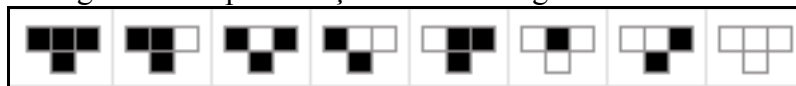
Figura 8 – Iterações da regra elementar 250



Fonte: adaptado de Wolfram (2002).

O padrão revelado entre as iterações apresentadas na Figura 8 pode ser formalizado por um conjunto de combinações possíveis entre uma célula qualquer deste espaço e suas células vizinhas imediatas, definindo também a nova cor a ser utilizada quando uma das combinações for identificada. A Figura 9 mostra a representação da regra elementar 250, onde a sequência de três células na linha superior de cada uma das oito transições define uma combinação possível entre as cores das células da vizinhança, sendo a célula imediatamente abaixo, a nova cor a ser utilizada na iteração seguinte.

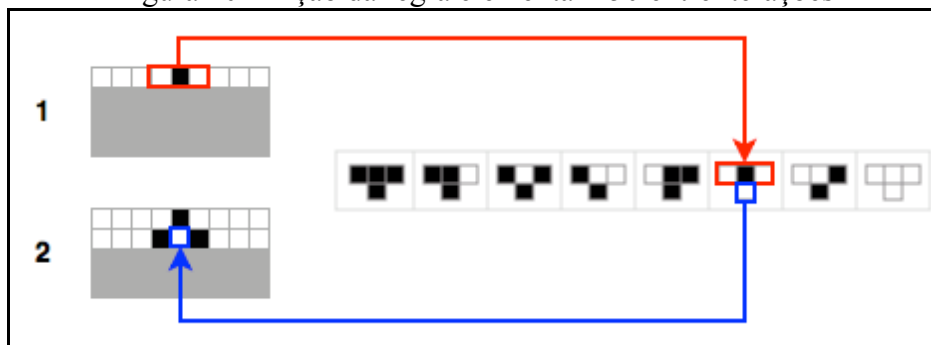
Figura 9 – Representação visual da regra elementar 250



Fonte: Wolfram (2002, p. 25).

Desta forma, é possível concluir que entre as iterações 1 e 2 apresentadas na Figura 8, foi utilizada a combinação da sexta transição (da esquerda para direita, na Figura 9) prevista pela regra, para definir que a célula de cor preta na iteração 1 tem cor branca na iteração 2. A Figura 10 mostra a aplicação da regra no espaço entre iterações 1 e 2 apresentadas na Figura 8.

Figura 10 – Ação da regra elementar 250 entre iterações

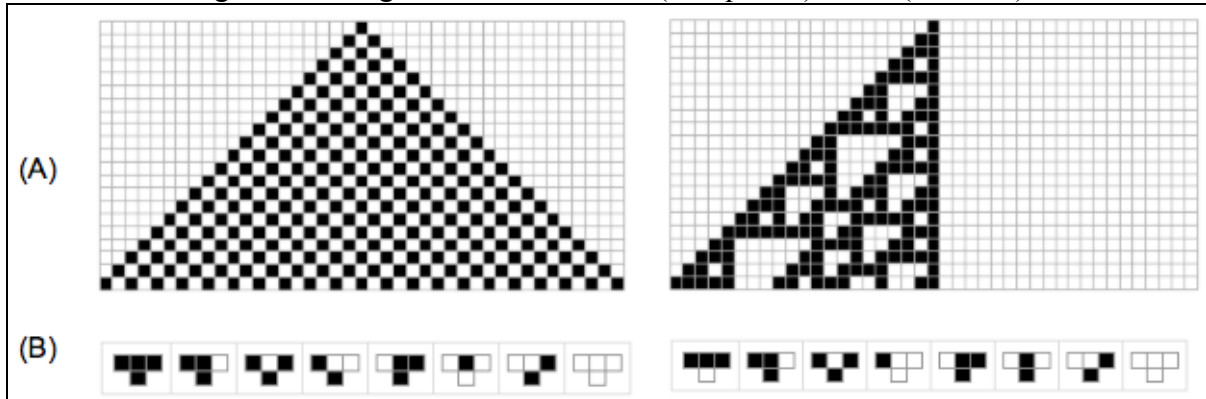


Fonte: adaptado de Wolfram (2002).

Ainda com base na Figura 10, é possível concluir que o mesmo mecanismo foi utilizado para colorir de preto as células vizinhas à célula indicada na segunda iteração, sendo a célula de cor preta à esquerda gerada a partir da sétima transição da regra, enquanto a célula de cor preta à direita gerada a partir da quarta transição.

A Figura 11 mostra vinte iterações das regras elementares 250 e 110 na parte superior (A), esquerda e direita, respectivamente, seguidas pela formalização de cada uma destas regras, na parte inferior (B).

Figura 11 – Regras elementares 250 (à esquerda) e 110 (à direita)



Fonte: adaptado de Wolfram (2002).

Os autômatos celulares unidimensionais apresentados na Figura 11 são do tipo elementar e, portanto, possuem apenas duas cores. O Apêndice A apresenta a evolução das 256 regras elementares a partir de uma condição inicial formada por apenas uma célula central de cor preta.

2.1.7 Condições de borda

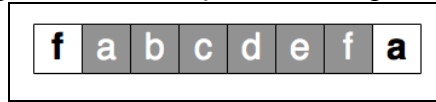
Conforme descrito por Schiff (2008), em muitos casos, o espaço celular será grande o suficiente em extensão para que seja considerado efetivamente infinito. Porém, em alguns casos, o espaço celular será limitado e, portanto, possuirá bordas. Nestes casos, conforme ainda previsto por Schiff (2008) e também por Floreano e Mattiussi (2008), as células próximas das bordas talvez não disponham de todas as células necessárias para a formação da vizinhança, comprometendo a execução do autômato celular.

Este problema pode ser resolvido através da definição de uma condição de borda adequada, que é capaz de tratar as células da borda de forma especial. A condição de borda é responsável por complementar a vizinhança da célula de referência localizada na borda do espaço celular, utilizando um comportamento pré-determinado. Segundo Schiff (2008), as três condições de borda mais comuns são: condição de borda periódica, condição de borda reflexiva e condição de borda fixa.

A condição de borda periódica (também conhecida como cíclica ou circular) considera que as extremidades do espaço celular são conectadas, sendo a primeira célula equivalente à célula imediatamente ao lado da última célula. A Figura 12 mostra a condição de borda

periódica, onde as células na cor cinza representam o espaço celular, as células de cor branca os vizinhos complementares e as letras em cada uma das células, seus respectivos estados.

Figura 12 – Condição de borda periódica

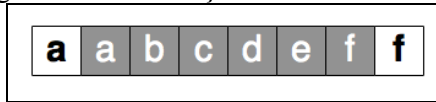


Fonte: adaptado de Schiff (2008).

A condição de borda periódica é a melhor forma de simular um espaço infinito, sendo que nenhuma borda é aparente para qualquer célula (SCHIFF, 2008). Esta condição de borda foi escolhida para a implementação do simulador proposto.

A condição de borda reflexiva repete a célula da borda em questão, complementando assim o vizinho faltante. A Figura 13 mostra a condição de borda reflexiva, onde as células na cor cinza representam o espaço celular, as células de cor branca os vizinhos complementares e as letras em cada uma das células, seus respectivos estados.

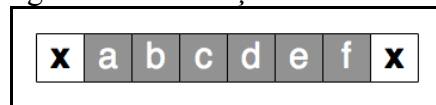
Figura 13 – Condição de borda reflexiva



Fonte: adaptado de Schiff (2008).

A condição de borda fixa assume que as células utilizadas para suprir a vizinhança de todas as células de borda possuirão um mesmo estado pré-definido. A Figura 14 mostra a condição de borda fixa, onde as células na cor cinza representam o espaço celular, as células de cor branca os vizinhos complementares e as letras em cada uma das células, seus respectivos estados.

Figura 14 – Condição de borda fixa



Fonte: adaptado de Schiff (2008).

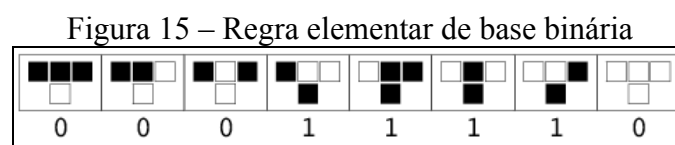
2.1.8 Cálculo de regras para autômatos celulares unidimensionais

As relações matemáticas envolvidas na morfologia dos autômatos celulares possuem relevância verificada no estudo de limites. Ao abordar os tipos de regra elementar, geral e totalista, é possível identificar que algumas relações se distribuem ao longo dos tipos de regra, porém, cada tipo possui também relações matemáticas exclusivas. A abordagem desta seção elabora as relações entre o número de cores em uma célula, o número de células em uma combinação, o número de combinações em uma regra e o número de regras em um tipo de regra.

Segundo Wolfram (1983), o número de combinações possíveis para uma regra elementar ou geral é equivalente ao número de cores elevado ao número de células envolvidas. Considerando o tipo de regra elementar, o número de cores envolvidas é restrito a exatamente 2, conforme definição, sendo o número de células envolvidas igual a 3, assim como nos tipos geral e totalista. Desta forma, obtém-se o cálculo $2^3=8$, resultando o número de combinações que estarão presentes em todas as regras elementares, conforme mostrado na Figura 9.

Conhecendo o número de combinações necessárias para que uma regra seja capaz de prever todas as situações do seu espaço, é possível determinar o número total de regras para este tipo de regra, assumindo o número de cores utilizado no cálculo das combinações como base e o número de combinações como expoente. Com esta definição, para as regras elementares tem-se $2^8=256$, sendo este o número total de regras elementares.

Algumas regras elementares foram anteriormente mencionadas através de um código, como as regras elementares 250 e 110, mostradas na Figura 11. Este código é um identificador único da regra em seu tipo de regra, que pode variar no intervalo de regras previstas para o tipo, sendo este intervalo de 0 a 255 no tipo de regra elementar. O cálculo do código da regra é feito através da associação do número de cores envolvidas, que atua como base numérica, ao número de combinações da regra. Esta associação é feita considerando a sequência de valores definidos em cada combinação como àqueles a serem utilizados na próxima iteração do espaço. A Figura 15 apresenta uma regra elementar com sua respectiva sequência de valores dentro da base numérica definida pelo número de cores.



Fonte: Weisstein (2009).

Como o número de cores de uma regra do tipo elementar é equivalente a 2, o número 00011110 presente na Figura 15 representa um número de base binária, equivalente a 30 na base decimal, portanto sendo conhecida como regra elementar 30. Ainda na Figura 15, o número 0 foi atribuído para combinações que resultam em uma célula de cor branca e o número 1 para as combinações que resultam em uma célula de cor preta.

2.1.9 Aplicações para autômatos celulares

Existem pelo menos quatro motivações parcialmente sobrepostas para o estudo de autômatos celulares, ordenados de acordo com o nível crescente de significância teórica:

como poderosos motores de computação; como simuladores discretos de sistemas dinâmicos; como veículos conceituais para o estudo de formação de padrões e complexidade; como modelos originais de física fundamental (ILACHINSKI, 2001).

Da mesma forma, assim como citado anteriormente, Wolfram (2002) apresenta mais de vinte áreas do conhecimento diferentes na qual os autômatos celulares podem ser aplicados, abordando a utilização e o potencial dos autômatos celulares nos espaços unidimensional e bidimensional.

2.1.9.1 Espaço unidimensional

A regra elementar de número 110, utilizada ao longo deste trabalho como exemplo de autômato celular unidimensional, é conhecida por apresentar um comportamento de equilíbrio entre ordem e caos (WOLFRAM, 2002). Possui uma propriedade até então única entre as 256 possíveis regras elementares: universalidade, ou seja, a capacidade de emular a máquina de Turing.

Conforme descrito por Cook (2008), a regra 110 é um autômato celular que executa repetidas atualizações simultâneas de uma linha infinita de valores binários. Os valores são atualizados da seguinte maneira: 0s são alterados para 1s em todas as posições onde o valor à direita é 1, enquanto 1s são alterados para 0s em todas as posições onde os valores à esquerda e à direita são ambos iguais a 1. Embora trivial para definir, o comportamento exibido pela regra 110 é surpreendentemente complexo e, conforme demonstrado em Cook (2004), ela é capaz de emular a atividade de uma máquina de Turing através da codificação da máquina e sua fita em um padrão repetido à esquerda, um padrão central e um padrão repetido à direita, na qual a regra 110 então atua.

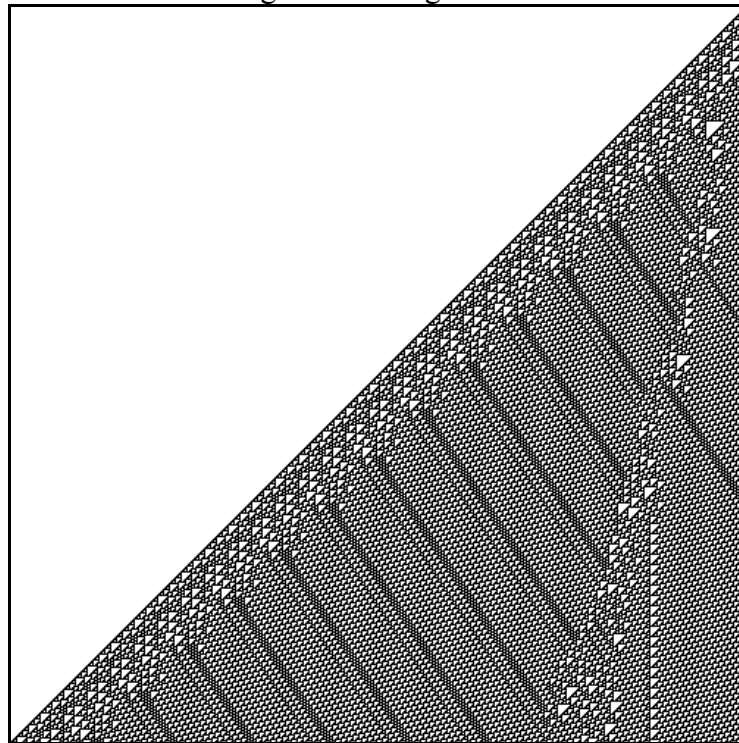
Wolfram (2002) elabora que os computadores práticos e as linguagens de programação têm sido tradicionalmente os únicos exemplos comuns de universalidade já encontrados. Considerando o fato de que estes tipos de sistema tendem a ser bastante complicados em sua construção, a intuição geral formada é de que qualquer sistema que consiga ser universal necessita, de alguma forma, ser baseado em regras subjacentes bastante complicadas.

Analisando o significado da universalidade na regra 110, Wolfram (2002) conclui que a implicação mais importante é a sugestão de que a universalidade seja um fenômeno imensamente mais comum, podendo ser verificada de forma ampla fora de sistemas construídos especificamente para este propósito, inclusive naqueles baseados em regras simples. A Figura 16 apresenta 500 iterações da regra elementar 110, criadas a partir de uma condição inicial de 500 células, onde apenas a última célula à direita possui a cor preta. Nesta

figura, é possível observar um padrão regular formado à esquerda, que se manifesta com frequência ao encontro de um padrão irregular à direita, sendo esta característica preservada ao longo da evolução da regra.

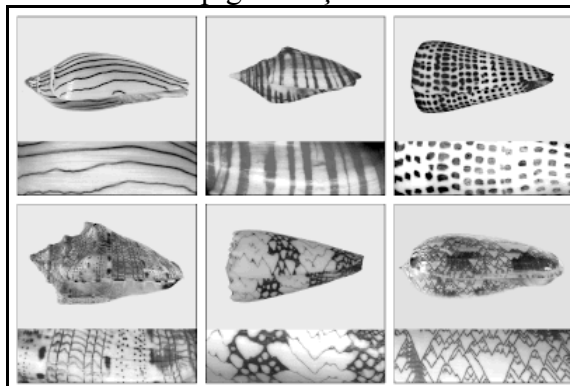
Considerando ainda os padrões revelados pelos autômatos celulares unidimensionais, Wolfram (2002) apresenta as similaridades encontradas em padrões biológicos de pigmentação e suspeita que pode ser a manifestação natural de regras extremamente simples. A Figura 17 apresenta exemplos de padrões de pigmentação encontrados em conchas de moluscos. Exemplos adicionais de pigmentação biológica e padrões naturais são apresentados no Apêndice B.

Figura 16 – Regra 110



Fonte: elaborado pelo autor.

Figura 17 – Padrões de pigmentação em conchas de moluscos

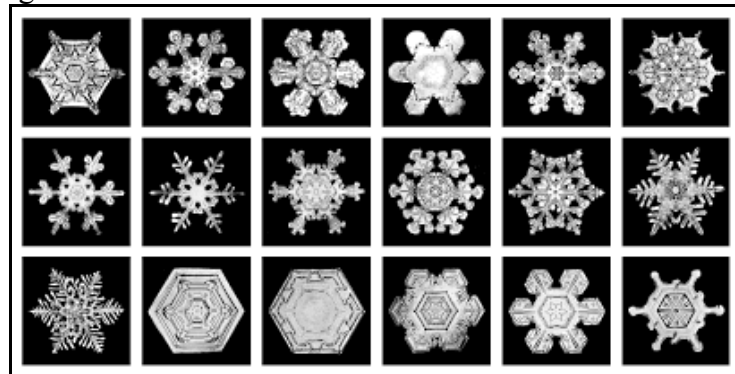


Fonte: Wolfram (2002, p. 423).

2.1.9.2 Espaço bidimensional

Ao considerar autômatos celulares de espaço bidimensional, é possível simular e estudar uma nova série de fenômenos como a formação de cristais e a turbulência em fluidos. Em escala microscópica, os cristais consistem de matrizes regulares de átomos dispostos da mesma forma que as células de um autômato celular. Um cristal é formado quando um líquido ou gás é resfriado abaixo de sua temperatura de congelamento. Os cristais sempre iniciam a partir de um componente externo – geralmente um grão de areia – e então, crescem progressivamente adicionando mais átomos à sua superfície (WOLFRAM, 2002). Como um exemplo de cristal, a Figura 18 apresenta flocos de neve com formas e tamanhos diversos.

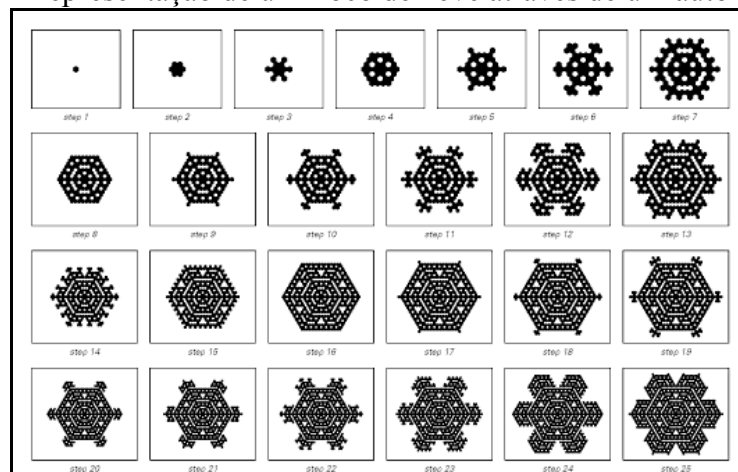
Figura 18 – Flocos de neve de formas e tamanhos diferentes



Fonte: Wolfram (2002, p. 370)

Para a representação da formação de um floco de neve, a Figura 19 mostra um autômato celular bidimensional que utiliza um espaço celular hexagonal. A condição inicial deste autômato celular é uma célula única de cor preta, que ao longo de vinte e cinco iterações é atualizada pela regra do autômato, expandindo o espaço celular utilizado de forma omnidirecional. O padrão resultante obtido em diferentes iterações é notavelmente similar a muitos flocos de neve reais (WOLFRAM, 2002).

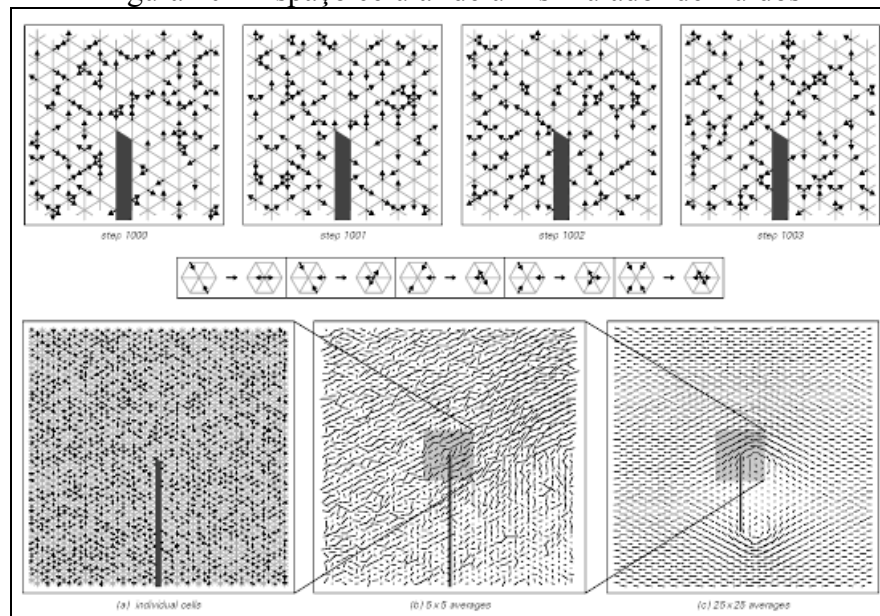
Figura 19 – Representação de um floco de neve através de um autômato celular



Fonte: adaptado de Wolfram (2002, p. 371).

Diferente dos cristais, fluidos físicos consistem de um grande número de moléculas em movimento, colidindo entre si (WOLFRAM, 2002). Desta forma, o comportamento de um fluido também pode ser representado através de um autômato celular bidimensional. A primeira sequência de quadrados horizontais da Figura 20 mostra um autômato celular bidimensional com um agrupamento de setas, responsáveis por indicar o sentido de movimento, e também, uma barra escura que representa um obstáculo sólido. Cada um dos quadrados representa uma iteração do espaço celular, onde é aplicada a regra representada pela sequência de 5 retângulos no centro da imagem. A última sequência de quadrados representa um afastamento do espaço celular, feito gradativamente (da esquerda para direita) através de uma média entre os estados das células localizadas naquela área.

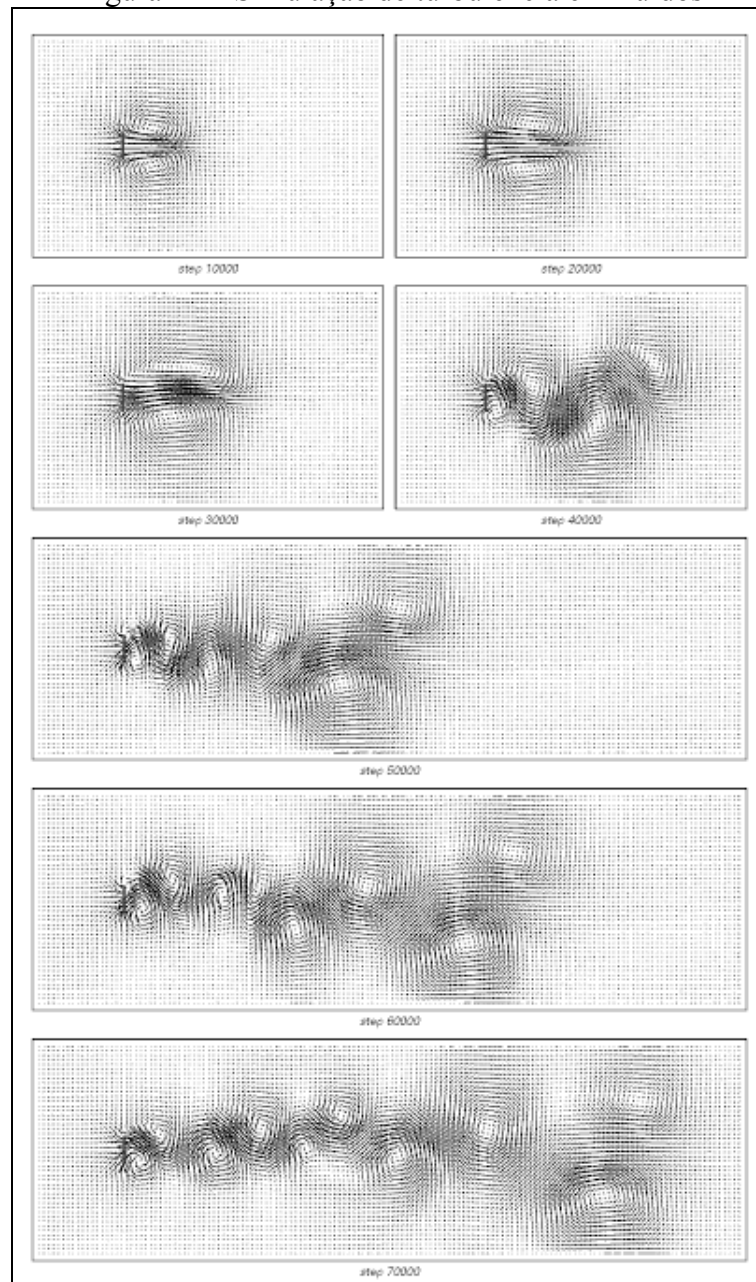
Figura 20 – Espaço celular de um simulador de fluidos



Fonte: adaptado de Wolfram (2002).

Com o aumento do número de iterações deste autômato celular, é possível perceber a formação de padrões correspondentes ao de um fluido. A Figura 21 apresenta uma sequência de 70.000 iterações deste autômato celular (10.000 por quadro) onde é possível perceber que o sentido de deslocamento do fluido ocorre da esquerda para direita, ocasionando uma área de turbulência à medida que entra em contato com o obstáculo sólido posicionado à esquerda do espaço celular. O Apêndice C apresenta exemplos de padrões e turbulência em fluidos.

Figura 21 – Simulação de turbulência em fluidos



Fonte: Wolfram (2002, p. 380).

2.2 SIMULAÇÃO

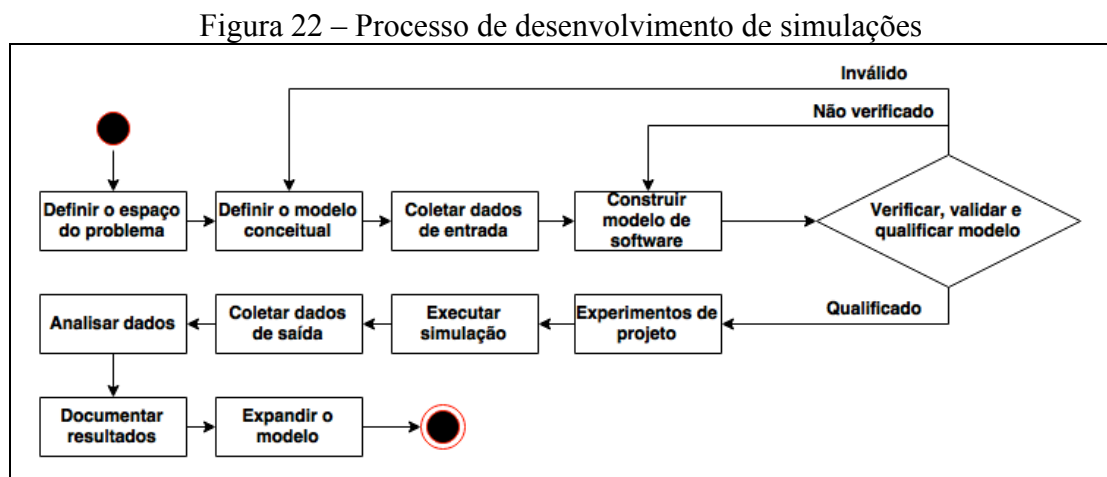
Conforme definido por Banks et al. (2005), uma simulação é a imitação da operação de um processo ou sistema do mundo real ao longo do tempo. Mesmo feita à mão ou em um computador, a simulação depende da criação e observação de uma história artificial de um sistema para criar inferências relacionadas às características operacionais do sistema real.

Banks et al. (2005) descreve os principais componentes de uma simulação, como o modelo do sistema a ser simulado, abordando também as situações apropriadas ou não para o uso de simulação. Como situações apropriadas para simulação destacam-se: o estudo e

experimentação das operações internas de um sistema complexo, uso pedagógico, melhorias de um sistema real com base no conhecimento adquirido ao construir o modelo de simulação, teste de condições diversas sem riscos ao sistema real, entre outros. O uso de simulação não é apropriado quando: é possível obter a solução do problema através de senso comum, a execução direta de um experimento é mais simples, o tempo de modelagem supera o limite previsto para a solução do problema, o custo da simulação é maior que a economia gerada pela solução do problema.

Como vantagens do uso de simulação, Banks et al. (2005) destaca: o teste de hipóteses e circunstâncias que envolvam o sistema, a compressão e a expansão do tempo para acelerar ou desacelerar o fenômeno sob investigação, a compreensão da relevância das variáveis envolvidas, a redução de riscos ao sistema real, entre outros. Como desvantagem, é apresentada a dificuldade em criar um modelo eficiente do sistema real que aborde seus aspectos de forma integral, ou a maior parte deles.

Tendo em vista os conceitos apresentados por Banks et al. (2005), o processo de construção de uma simulação pode ser estudado com maior domínio das implicações preliminares. Smith (1998) apresenta um processo definido para o desenvolvimento, validação, operação e análise dos resultados de uma simulação. Este processo é mostrado pela Figura 22 através de um diagrama de atividades.

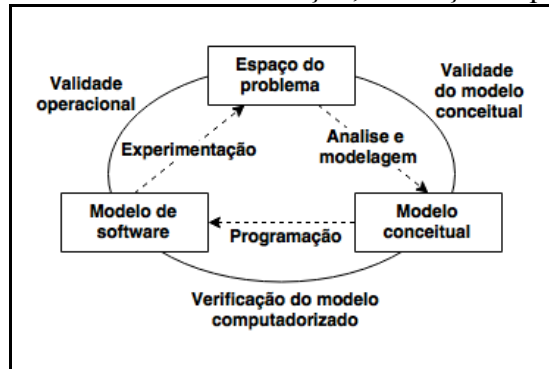


Fonte: adaptado de Smith (1998).

Para a definição do espaço do problema, Smith (1998) explica que deve ser feita a definição correta do problema a ser abordado pelo modelo, junto à definição dos limites entre o problema e o ambiente no qual ele está inserido. Para a definição do modelo conceitual, deve ser definido o algoritmo a ser utilizado, bem como os dados de entrada e dados de saída. Com estes aspectos definidos, os dados de entrada necessários para operar e definir o modelo são coletados, e em seguida, é construído o modelo de software responsável por executar a

simulação, seguindo os princípios de engenharia de software. A fase de verificação, validação e qualificação é essencial para garantir que os algoritmos do modelo, os dados de entrada e demais diretivas estão corretas e solucionam o problema identificado no início do processo. A Figura 23 apresenta em detalhe como funciona esta etapa.

Figura 23 – Processo de verificação, validação e qualificação



Fonte: adaptado de Smith (1998).

Após a verificação, validação e qualificação, são executados os experimentos de projeto, onde são identificados os métodos de maior precisão para executar a simulação, o que torna a simulação apta para a execução. Com a execução da simulação, é efetuada a coleta dos dados de saída e iniciado o processo de análise, onde os dados são organizados e processados a fim de responder as perguntas que motivaram a construção do simulador. Por fim, os resultados são documentados e o modelo podem ser reavaliados e expandidos, a fim de executar simulações com propósitos diferentes, quando necessário.

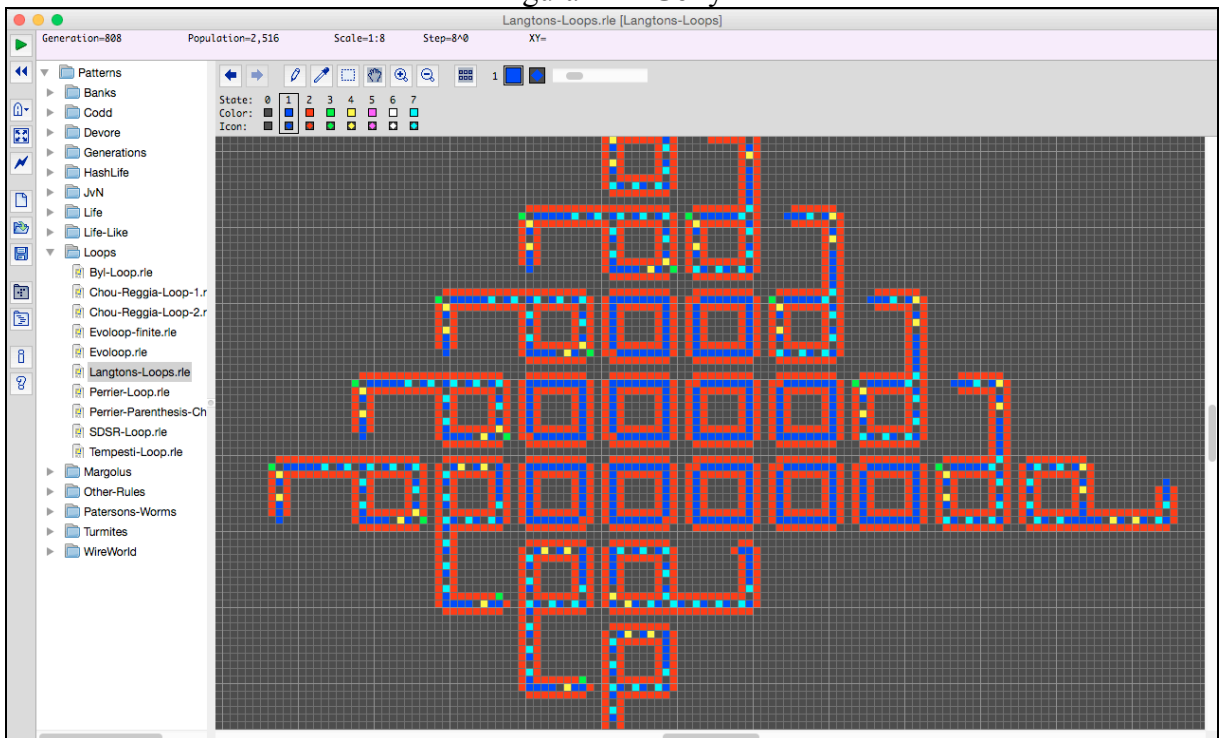
2.3 TRABALHOS CORRELATOS

A seguir estão relacionados dois trabalhos correlatos ao proposto: o Golly (ROKICKI et al., 2015), ferramenta *open source* para explorar autômatos celulares, e o Mathematica (WOLFRAM ALPHA LLC, 2016), um programa de álgebra computacional.

2.3.1 Golly

Golly, segundo Rokicki et al. (2015), é uma aplicação *open source* e multi-plataforma para explorar autômatos celulares em espaços bidimensionais como Game of Life, assim como os autômatos celulares elementares. Golly acompanha uma série de exemplos de autômatos e *scripts* que podem ser executados e modificados pela própria ferramenta. Os autômatos celulares podem ser executados através de um conjunto específico de algoritmos como QuickLife, HashLife, Generations, JvN e RuleLoader, com suporte a mais de 256 estados por célula. A Figura 24 mostra a execução do autômato celular bidimensional conhecido como Langston's Loop, através da ferramenta Golly.

Figura 24 – Golly

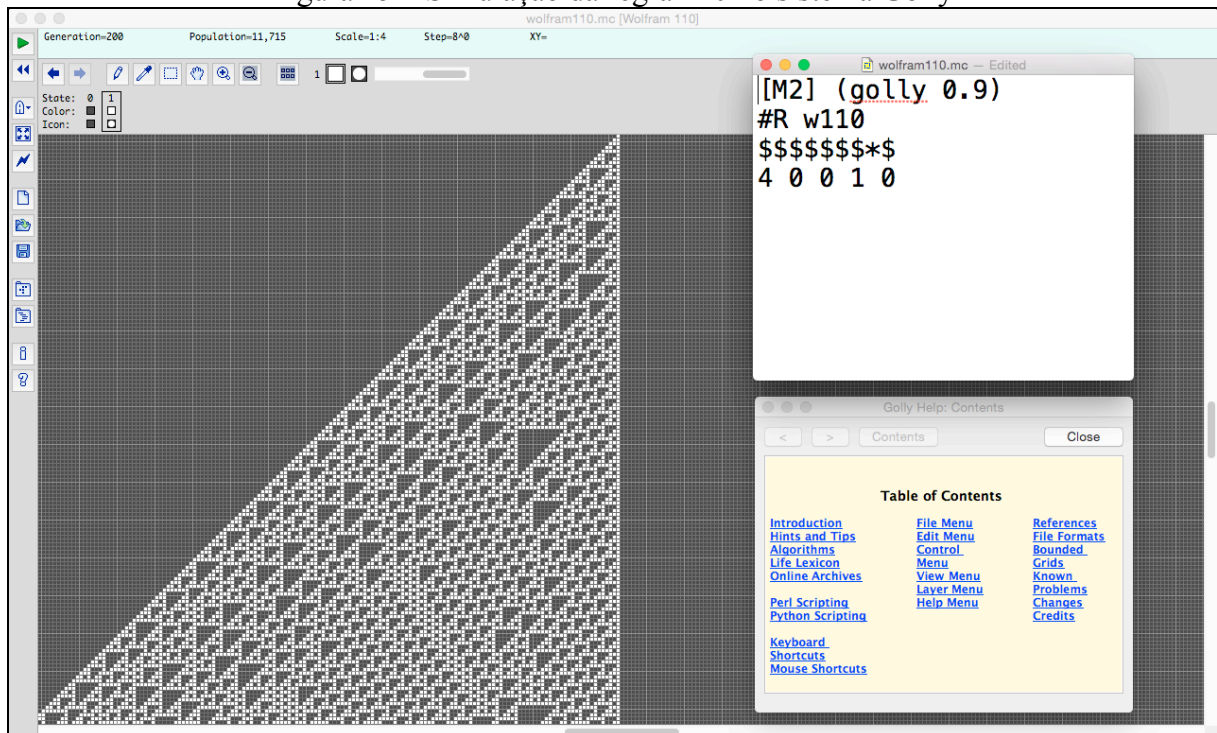


Fonte: elaborado pelo autor.

Este simulador é considerado por seus usuários como uma das ferramentas *open source* mais completa e de maior performance para trabalhar com autômatos celulares (ROKICKI et al., 2015). Porém, apesar de oferecer recursos de edição para os parâmetros de entrada e controle sobre a execução do autômato, Golly não oferece recursos didáticos que contribuam ao ensino do funcionamento dos autômatos celulares, apenas uma documentação breve e específica com uma abordagem para aqueles que já conhecem a área de nível intermediário a avançado.

A Figura 25 mostra a execução da regra 110 no Golly (à esquerda), o arquivo criado para executar a simulação da regra (à direita, superior) e o conteúdo da ajuda oferecida pelo software (à direita, inferior). O arquivo criado para efetuar a simulação da regra possui extensão `.mc` (formato Macrocell criado para o Golly), onde a primeira linha representa o cabeçalho obrigatório do arquivo, a segunda linha especifica o número da regra a ser computada precedida pelo prefixo `#R w`, onde `#R` representa “rule” e `w` representa “Wolfram”, sendo apenas as regras elementares de número par atualmente suportadas. As duas últimas linhas do arquivo são a configuração da condição inicial e sua disposição no espaço celular. Este arquivo precisa ser executado utilizando o algoritmo QuickLife selecionado na barra de ferramentas localizada à esquerda da janela do sistema.

Figura 25 – Simulação da regra 110 no sistema Golly



Fonte: elaborado pelo autor.

Golly possui recursos de interação visual e controle da velocidade de execução que auxiliam durante o progresso da simulação, porém, a configuração da regra não é exibida para o usuário, dificultando a compreensão sobre a origem do padrão visualizado. Além disso, a criação de um arquivo de sintaxe própria e a escolha do algoritmo correto para a simulação também contribuem para qualificar a experiência do usuário como não-intuitiva.

2.3.2 Mathematica

Mathematica é, segundo Wolfram Alpha LLC (2016), uma ferramenta para computação matemática simbólica, que utiliza a linguagem multi-paradigma Wolfram, criada sob a influência e necessidades de um método de interação formal com a plataforma.

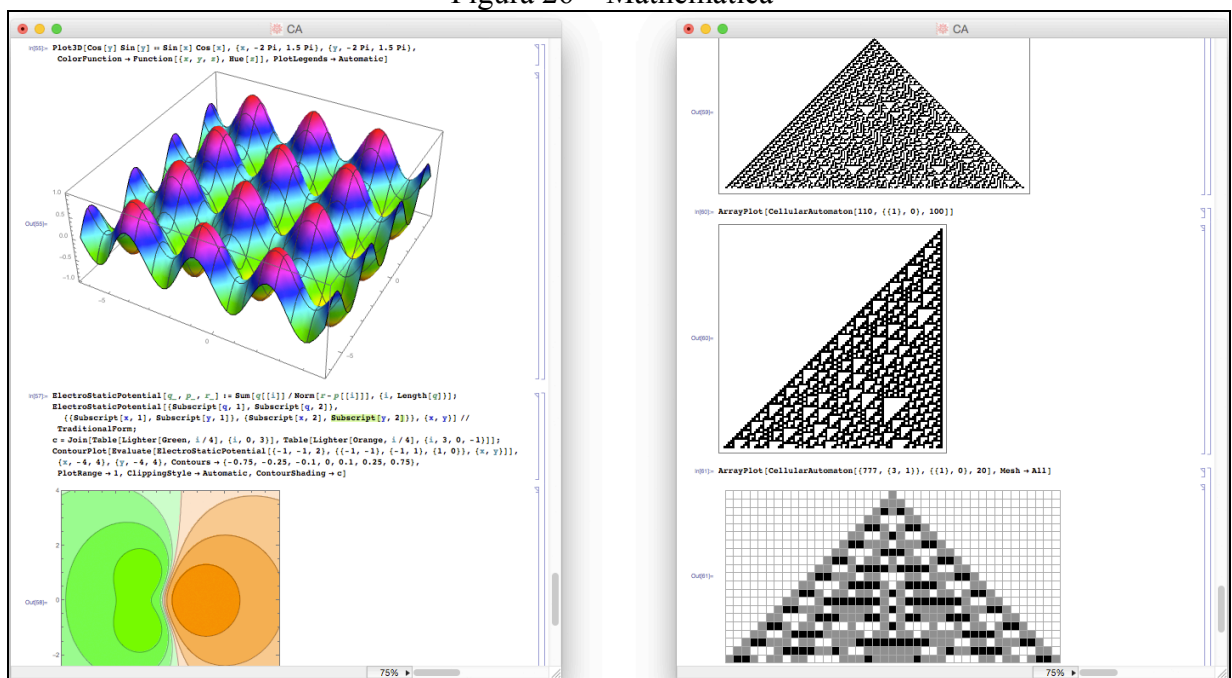
A interação com este recurso é feita através de uma interface de características similares à de um terminal de comandos, onde a entrada é feita utilizando a linguagem Wolfram e o resultado da computação pode ser apresentado de forma dinâmica, através de um conjunto de dados naturais, plotagens bidimensionais ou tridimensionais, entre outros.

O software Mathematica é utilizado por diferentes áreas científicas para realizar uma variedade de operações de alta complexidade matemática, computacional e de engenharia. Entre as operações suportadas por Mathematica, está a computação dos autômatos celulares conforme os conceitos definidos por Wolfram (1983), permitindo integrar esta operação a outros recursos da plataforma, de acordo com a necessidade e resultados esperados.

Mathematica suporta os espaços unidimensional, bidimensional e tridimensional, possibilitando a utilização dos tipos de regra elementar, geral, totalista e totalista exterior (não abordados na revisão bibliográfica). A configuração inicial dos autômatos é informada utilizando a linguagem Wolfram, onde é possível definir parâmetros adicionais para determinar como deve ser efetuada a computação do autômato em questão, permitindo assim, obter sua execução passo-a-passo. Por ser uma ferramenta científica, Mathematica não possui recursos didáticos nativos que apresentam o funcionamento dos autômatos celulares, mas possui uma documentação de qualidade que orienta o processo experimental daqueles que iniciam seus estudos na área.

A Figura 26 mostra duas janelas do software Mathematica, sendo a janela do lado esquerdo com a computação de operações matemáticas e a janela do lado direito com a computação de alguns autômatos celulares unidimensionais.

Figura 26 – Mathematica



Fonte: elaborado pelo autor.

Conhecer a linguagem Wolfram é fundamental para usar a ferramenta, que apesar de não ser difícil, pode impactar na intuição de alguns usuários. A Figura 27 apresenta duas computações da regra elementar 110. Nessa figura, o primeiro comando executado é `CellularAutomaton[110, {{1}, 0}, 20]`, onde `CellularAutomaton` corresponde à função de computação de autômatos celulares, 110 informa o número da regra elementar, `{{1}, 0}` define a condição inicial e 20 define a quantidade de iterações do espaço celular. O resultado bruto desta operação é uma matriz de inteiros que representam os estados de cada uma das células do espaço celular, onde 0 é equivalente à cor branca e 1 equivalente à cor

3 DESENVOLVIMENTO

O simulador desenvolvido foi nomeado CAS, um acrônimo de Cellular Automata Simulator, ou seja, simulador de autômatos celulares. Durante o desenvolvimento do CAS, adotou-se uma perspectiva de abstração com potencial para computar não só os autômatos celulares unidimensionais, mas também, autômatos celulares multidimensionais. A partir do estudo dos diferentes tipos de autômatos celulares, separou-se as características gerais das específicas a fim de que sejam aplicáveis a qualquer tipo de autômato celular. Sendo assim, esta perspectiva abstrata deve ser considerada durante a apresentação do desenvolvimento do simulador, feita ao longo deste capítulo, que está organizado em quatro seções: requisitos, especificação, implementação e análise dos resultados.

3.1 REQUISITOS

O simulador de autômatos celulares unidimensionais deverá:

- a) permitir ao usuário informar a configuração da regra a ser computada (Requisito Funcional - RF);
- b) permitir ao usuário informar o número de iterações a serem computadas (RF);
- c) permitir ao usuário informar a configuração inicial do autômato celular (RF);
- d) permitir ao usuário informar se a execução será passo-a-passo ou automática (RF);
- e) executar a computação do autômato celular passo-a-passo com a intervenção do usuário (RF);
- f) executar a computação do autômato celular de forma automática (RF);
- g) executar a computação de autômatos celulares elementares (RF);
- h) exibir a representação gráfica da computação do autômato celular (RF);
- i) permitir ao usuário salvar a configuração de um autômato celular (RF);
- j) permitir ao usuário carregar a configuração de um autômato celular a partir de um arquivo salvo anteriormente (RF);
- k) permitir ao usuário salvar o resultado da computação de um autômato celular como um arquivo de imagem (RF);
- l) utilizar a linguagem Java (Requisito Não Funcional - RNF);
- m) utilizar a biblioteca Processing para renderizar a representação visual da simulação (RNF);
- n) possuir testes unitários desenvolvidos para a ferramenta JUnit (RNF).

3.2 ESPECIFICAÇÃO

Nesta seção é apresentada a especificação do simulador. Tem-se a estrutura do projeto com classes agrupadas em módulos, seguida pela descrição dos componentes em cada um dos módulos.

3.2.1 Estrutura do projeto

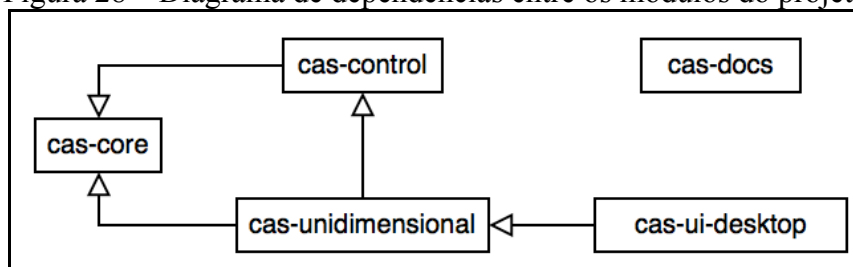
O projeto está organizado em cinco módulos: *cas-core*, *cas-unidimensional*, *cas-control*, *cas-ui-desktop* e *cas-docs*. Cada um destes módulos possui recursos específicos, que podem ser compartilhados com outros módulos através de uma relação de dependência. O Quadro 1 relaciona o nome e a descrição de cada um dos módulos do projeto, enquanto a Figura 28 apresenta o diagrama de dependências entre os módulos.

Quadro 1 – Descrição dos módulos do projeto

Módulo	Descrição
<i>cas-core</i>	estruturas abstratas que representam as características comuns de todos os tipos de autômatos celulares, independente do número de dimensões
<i>cas-control</i>	recursos abstratos para o controle do ciclo de vida da simulação
<i>cas-unidimensional</i>	implementação unidimensional dos módulos <i>cas-core</i> e <i>cas-control</i> , responsável por definir características específicas e por controlar a simulação de autômatos celulares unidimensionais
<i>cas-ui-desktop</i>	componentes de interface em padrão desktop, incluindo aqueles necessários para a renderização da simulação
<i>cas-docs</i>	documentação

Fonte: elaborado pelo autor.

Figura 28 – Diagrama de dependências entre os módulos do projeto



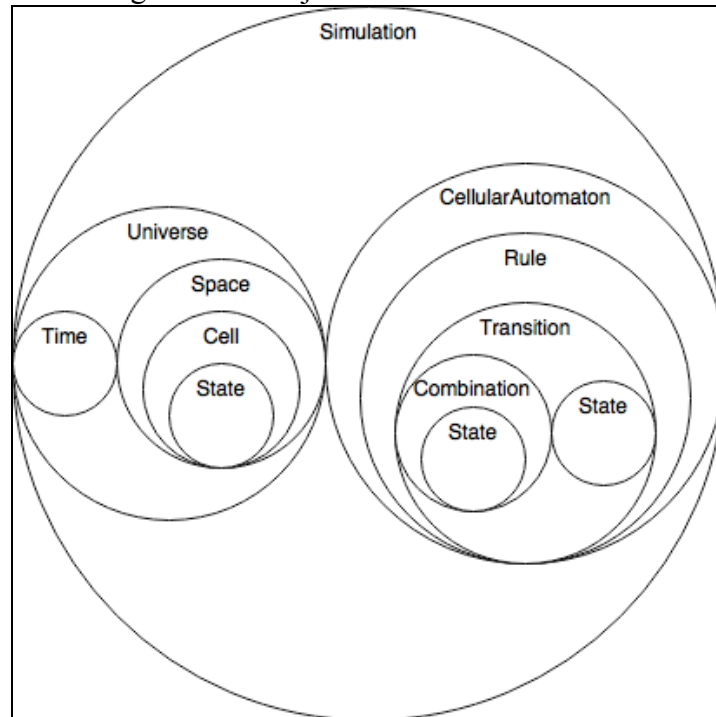
Fonte: elaborado pelo autor.

3.2.2 Módulo *cas-core*

Conforme a descrição apresentada no Quadro 1, todas as classes contidas no módulo *cas-core* são abstratas e representam o modelo de dados e a lógica de funcionamento de todos os tipos de autômatos celulares, em alto nível. Para a especificação deste módulo, foram estudados os diferentes tipos de autômatos celulares, identificando as características comuns e as características específicas de cada um.

Foi formalizado um modelo de dados com as características em comum entre os autômatos celulares, capaz de manifestar o padrão de comportamento identificado entre eles. Para uma visualização simplificada do modelo de dados utilizado no módulo `cas-core`, foi criado o diagrama de conjuntos mostrado na Figura 29, que expressa a composição entre cada uma das classes do núcleo, desconsiderando a cardinalidade.

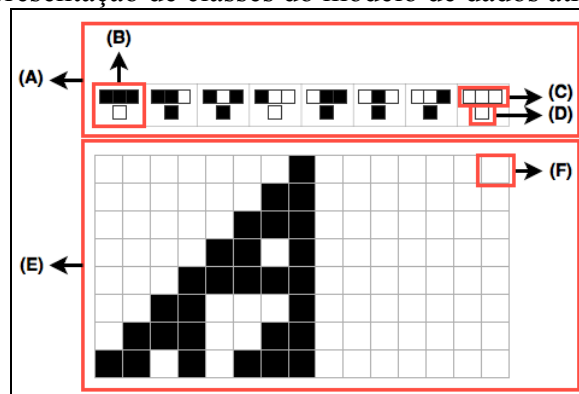
Figura 29 – Diagrama de conjuntos das classes do módulo `cas-core`



Fonte: elaborado pelo autor.

Os nomes das classes do modelo foram escolhidos de forma a manterem similaridade com a terminologia apresentada no capítulo anterior. Algumas das classes estão diretamente relacionadas com as imagens presentes em Wolfram (2002), conforme apresentado na Figura 30. Nesta figura, foram destacadas as classes `Rule` (A), `Transition` (B), `Combination` (C), `State` (D), `Space` (E) e `Cell` (F), sob a tradicional representação da regra elementar 110.

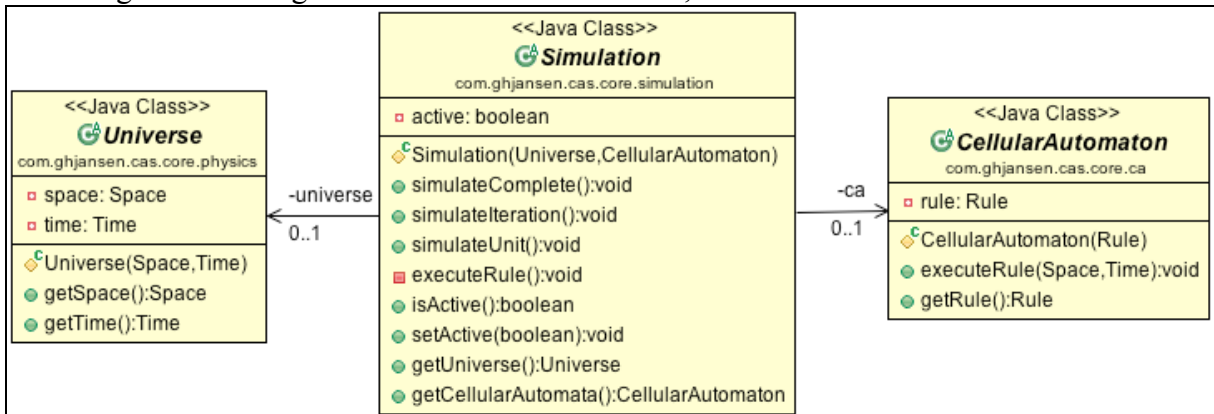
Figura 30 – Representação de classes do modelo de dados através da regra 110



Fonte: elaborado pelo autor.

Como pode ser visualizado ainda na Figura 29, a classe `Simulation` contém as demais classes do modelo, formando uma relação direta com as classes `Universe` e `CellularAutomaton`. O diagrama de classes correspondente ao relacionamento entre as classes `Simulation`, `Universe` e `CellularAutomaton` pode ser visualizado na Figura 31.

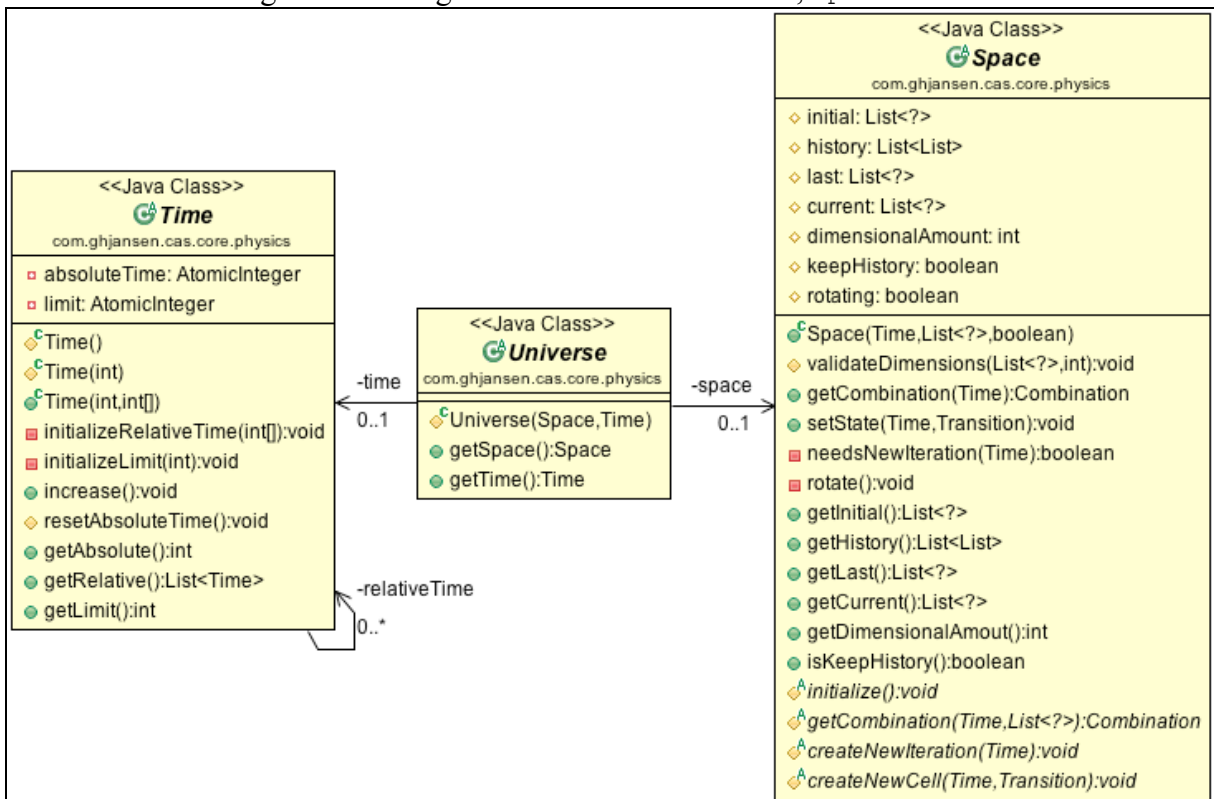
Figura 31 – Diagrama de classes: `Simulation`, `Universe` e `CellularAutomaton`



Fonte: elaborado pelo autor.

A classe `Universe` é uma representação simbólica das características de aspecto físico utilizadas pelo autômato celular durante a simulação, sendo estas características o espaço e o tempo, especificadas através dos atributos `space` e `time`, respectivamente. O diagrama apresentado pela Figura 32 contém as classes `Universe`, `Space` e `Time`.

Figura 32 – Diagrama de classes: `Universe`, `Space` e `Time`



Fonte: elaborado pelo autor.

Devido à complexidade e importância de algumas das demais classes do modelo de dados, estas são apresentadas nas próximas seções em uma sequência compatível com a ordem de relevância verificada no funcionamento do simulador.

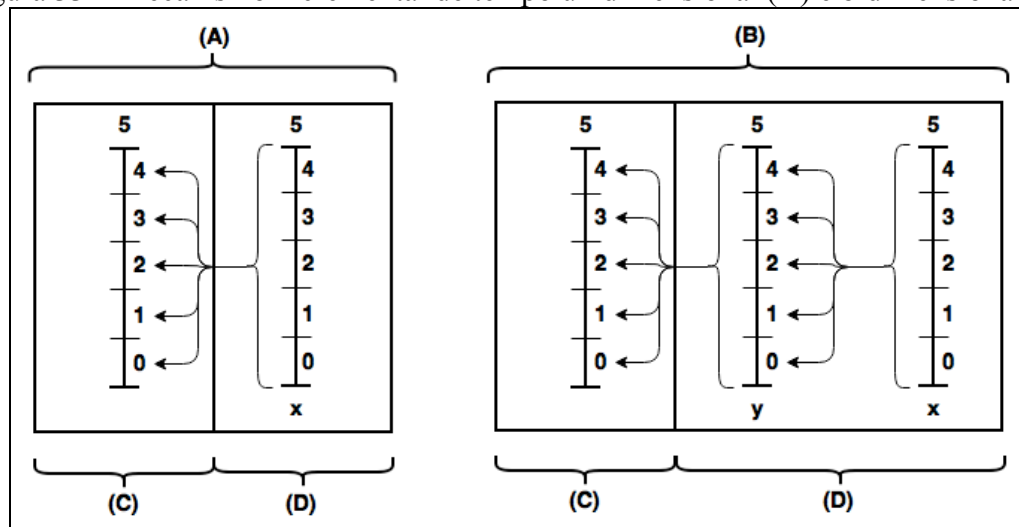
3.2.2.1 Classe `Time`

A classe `Time` é responsável por gerenciar um dos dados mais importantes durante a simulação: o tempo. O tempo é uma informação fundamental para a localização e gerenciamento de células, atua como o guia do autômato celular dentro do espaço ao longo do processamento da simulação. O tempo e o espaço são dois conceitos intimamente relacionados pelo aspecto dimensional, ou seja, o número de dimensões utilizadas pelo autômato celular afeta não só o espaço celular, mas também o tempo. Esta característica é orientada à natureza sequencial do simulador, que atualiza o espaço e o tempo em uma operação atômica consecutivamente até atingir os limites parametrizados para a simulação.

Para permitir que o tempo opere de forma dimensional, considerou-se o tempo absoluto e o tempo relativo. O tempo absoluto é responsável por manter a quantidade de iterações do espaço já processadas pelo autômato celular e pode ser representado através de um número inteiro. O tempo relativo informa qual a próxima célula a ser processada dentro da iteração atual e é representado por uma lista de números inteiros, onde o número de elementos da lista é igual a quantidade de dimensões do autômato celular.

Desta forma, é correto afirmar que a classe `Time` é um contador dinâmico incrementado a partir de seu limite inferior (zero) atuando sobre cada uma das dimensões (tempo relativo) até processar completamente uma iteração de espaço (tempo absoluto), repetindo este ciclo até atingir os limites de tempo. A Figura 33 ilustra o mecanismo incremental para os tempos unidimensional (A) e bidimensional (B) com seus respectivos tempos absolutos (C) e relativos (D). Neste exemplo, o limite de todos os contadores é igual a 5 e, tanto para o tempo unidimensional (A) quanto para o tempo bidimensional (B), a ação incremental ocorre da direita para a esquerda. Quando um contador de tempo relativo (D) atinge o seu limite, é incrementada uma unidade no contador imediatamente à esquerda, sendo o seu limite inferior restaurado. Este processo se repete até incrementar o tempo absoluto (C) até o seu limite, o que representa o fim da simulação.

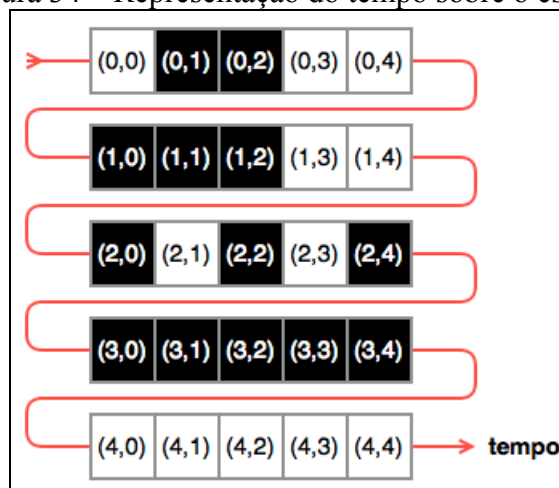
Figura 33 – Mecanismo incremental de tempo unidimensional (A) e bidimensional (B)



Fonte: elaborado pelo autor.

Para uma representação mais concreta da ação incremental do tempo, a Figura 34 utiliza o espaço celular unidimensional com a regra elementar 110. Cada uma das células possui uma notação entre parênteses, onde o valor à esquerda da vírgula representa o tempo absoluto e o valor à direita da vírgula, o tempo relativo. A seta vermelha é uma representação do avanço do tempo sobre o espaço.

Figura 34 – Representação do tempo sobre o espaço



Fonte: elaborado pelo autor.

Ainda na Figura 34, é correto afirmar que cada linha de células representa uma iteração do espaço unidimensional, criadas ao incrementar o tempo absoluto, e cada coluna representa uma nova célula na iteração correspondente, criada ao incrementar o tempo relativo. Sendo os limites de tempo absoluto e relativo ambos definidos como 5, são produzidas cinco iterações de um espaço unidimensional de cinco células. Tanto as células quanto as iterações são identificadas por contadores que iniciam em 0 terminam em 4, evidenciando que os limites de tempo absoluto e relativo são limites exclusivos.

Matematicamente, é possível afirmar que a quantidade de vezes que a classe `Time` incrementa seus contadores durante uma simulação é equivalente ao produtório dos limites dos contadores relativos, multiplicado pelo limite do contador absoluto, menos um. Esta fórmula é apresentada na Figura 35, onde “a” representa o contador absoluto, “d” representa a quantidade de contadores relativos (ou quantidade de dimensões), “r_i” representa um contador relativo e “lim” é a função que obtém o limite de cada contador.

Figura 35 – Fórmula para cálculo do tempo

$$\Gamma = \text{lim}(a) \left(\prod_{i=1}^d \text{lim}(r_i) \right) - 1$$

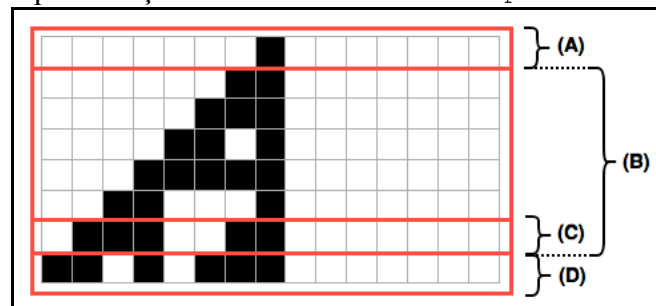
Fonte: elaborado pelo autor.

Com a fórmula apresentada pela Figura 35, é possível determinar que para um autômato celular unidimensional de 5 células com 5 iterações, a quantidade de vezes que a classe `Time` irá incrementar o tempo será de $5 * (5) - 1 = 24$. Da mesma forma, para um autômato celular bidimensional com 10 linhas, 20 colunas e 30 iterações, a execução incremental máxima será de $30 * (10 * 20) - 1 = 5999$.

3.2.2.2 Classe `Space`

A classe `space` é uma matriz dinâmica responsável por manter todas as células da condição inicial do espaço definido para a simulação (atributo `initial`), o histórico das iterações já processadas (atributo `history`), a última iteração processada (atributo `last`) e a iteração que encontra-se em processamento (atributo `current`). Os atributos `initial`, `last` e `current` são uma lista de tipo genérico a fim de permitir estruturas multidimensionais, já o atributo `history` é uma lista de listas, uma vez que `history` armazena cópias de `current`. A Figura 36 mostra uma representação dos atributos `initial` (A), `history` (B), `last` (C) e `current` (D), utilizando a computação da regra elementar 110.

Figura 36 – Representação dos atributos da classe `Space` através da regra 110



Fonte: elaborado pelo autor.

Em seu fluxo lógico, a classe `Space` utiliza a classe `Time` para obter uma combinação da condição inicial ou da última iteração, que é utilizada pelo autômato celular para determinar a transição responsável por definir o estado da próxima célula. Uma vez que o autômato celular encontra a transição correspondente à combinação, é criada uma nova célula utilizando a transição de origem, sendo assim, definido o estado da nova célula.

As células do espaço são representadas pela classe `Cell`, que possui um atributo da classe `State`, responsável por armazenar um valor inteiro correspondente ao estado da célula. Devido à complexidade no gerenciamento das células, que ocorre em função da quantidade variável de dimensões em autômatos celulares, algumas operações foram definidas na classe `Space` como métodos abstratos, pois permite formalizar a lógica geral de gerenciamento de espaço omitindo as especificidades dimensionais. Desta forma, torna-se responsabilidade da classe que estende a classe `Space` através de herança, definir como as operações abstratas devem acontecer para a quantidade de dimensões que a implementação de espaço aborda. O Quadro 2 relaciona os métodos abstratos da classe `Space` bem com uma descrição do que estes métodos devem implementar.

Quadro 2 – Métodos abstratos da classe `Space`

Método	Descrição
<code>initialize()</code>	cria novas listas para os atributos <code>history</code> e <code>current</code> , seguindo o padrão dimensional da implementação
<code>getCombination</code> (<code>Time</code> , <code>List</code>)	obtem uma combinação de estados do espaço <code>List</code> , com base na célula de referência indicada pelo valor de <code>Time</code> , considerando condições de borda
<code>createNewCell</code> (<code>Time</code> , <code>Transition</code>)	cria uma nova célula no espaço <code>current</code> , conforme o padrão dimensional da implementação, utilizando <code>Transition</code> para definir o estado na nova célula
<code>createNewIteration</code> (<code>Time</code>)	cria uma nova lista seguindo o padrão dimensional da implementação, associando-a ao atributo <code>current</code>

Fonte: elaborado pelo autor.

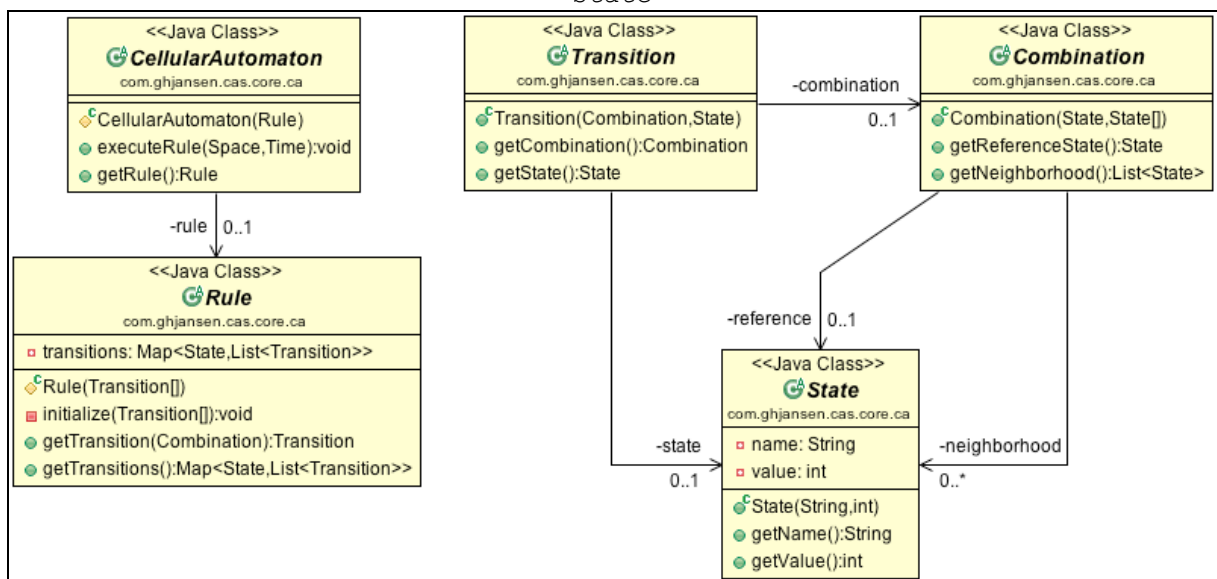
Entre as operações executadas pela classe `Space`, encontra-se a operação de rotação do espaço. A rotação do espaço ocorre quando a última célula da iteração atual é estabelecida. Nesta situação, o atributo `current` é clonado, adicionado à lista do atributo `history` e definido como atributo `last`. Após a execução deste procedimento, uma nova lista é criada para o atributo `current` através do método `createNewIteration`, conforme descrito no Quadro 2, e o processamento repete o mesmo ciclo até atingir os limites de tempo absoluto e relativo. Conclui-se que a classe `Space` não possui todos os detalhes necessários para operar espaços multidimensionais, mas sim, todas as características comuns entre os espaços com

diferentes dimensões. As características específicas são representadas através dos métodos abstratos, implementados por uma classe que estende a classe `Space`.

3.2.2.3 Classe `CellularAutomaton`

A classe `CellularAutomaton` é a representação conceitual do autômato celular utilizado na simulação. A lógica desta classe utiliza outras classes importantes como `Rule`, `Transition`, `Combination` e `State`, como pode ser observado através do diagrama de classes apresentado pela Figura 37.

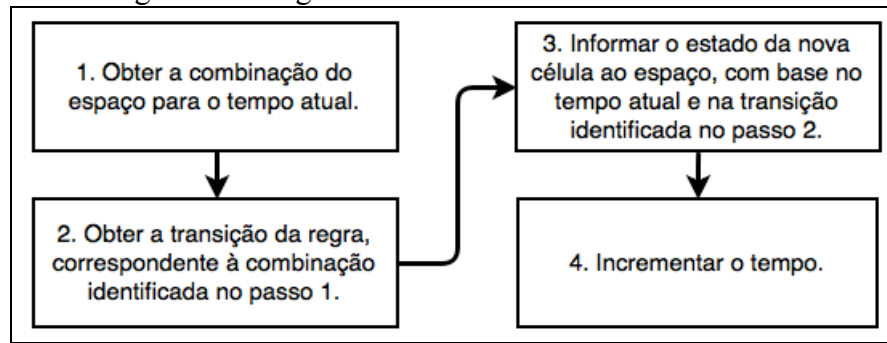
Figura 37 – Diagrama de classes: `CellularAutomaton`, `Rule`, `Transition`, `Combination` e `State`



Fonte: elaborado pelo autor.

A classe `State` possui um número inteiro responsável por definir o estado de uma célula. A classe `Combination` possui uma combinação de estados que representa uma possível vizinhança formada durante a simulação. A classe `Transition` é uma associação entre uma combinação e um novo estado, sendo este estado utilizado pelo autômato celular caso a combinação buscada pelo autômato seja equivalente a da transição.

A classe `Rule` representa a regra utilizada pelo autômato celular para identificar qual será o estado da nova célula, quando o espaço identificar uma combinação de estados específica na condição inicial ou em uma iteração anterior. A Figura 38 apresenta o algoritmo executado pelo autômato celular para definir o estado de uma nova célula.

Figura 38 – Algoritmo da classe `CellularAutomaton`

Fonte: elaborado pelo autor.

O algoritmo apresentado pela Figura 38 é executado repetidamente até que o limite de tempo seja atingido. A quantidade de execuções do algoritmo do autômato celular também possui uma relação matemática, semelhante à fórmula apresentada pela Figura 35 para o cálculo do tempo. A Figura 39 apresenta a fórmula para o cálculo da quantidade de execuções do algoritmo do autômato celular, onde “a” representa o contador de tempo absoluto, “d” representa a quantidade de contadores de tempos relativos (ou quantidade de dimensões), “r_i” representa um contador relativo e “lim” é a função que obtém o limite de cada contador.

Figura 39 – Fórmula para cálculo da quantidade de execuções da regra

$$R = \text{lim}(a) \left(\prod_{i=1}^d \text{lim}(r_i) \right)$$

Fonte: elaborado pelo autor.

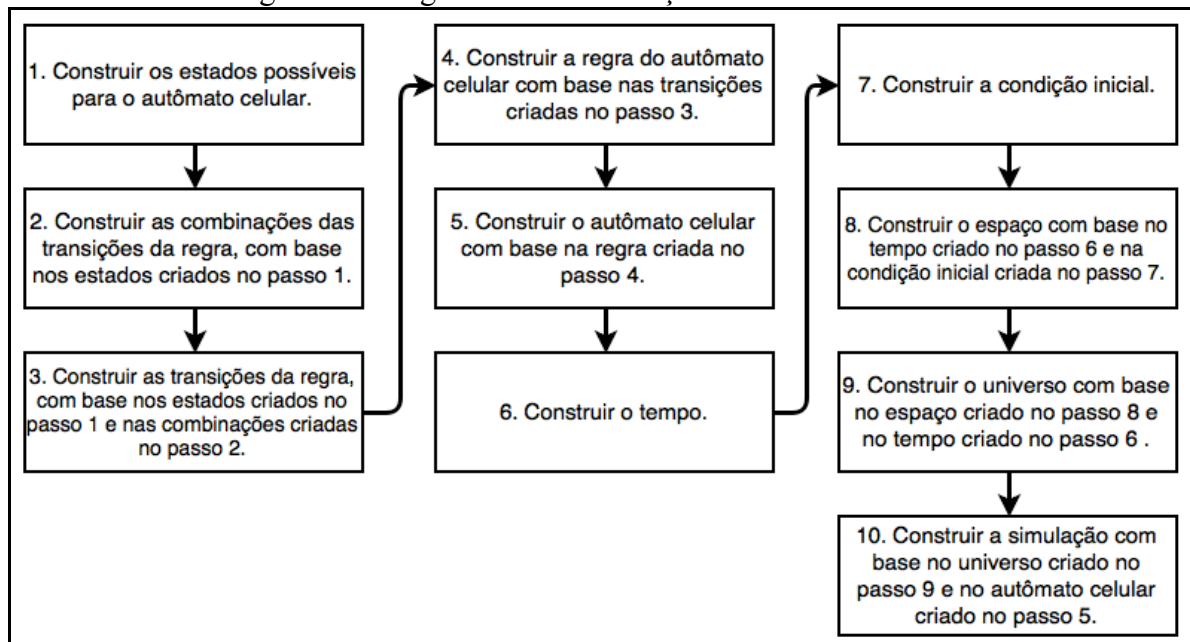
Como pode ser observado nas fórmulas apresentadas na Figura 35 e na Figura 39, a diferença entre as suas relações matemáticas é a subtração do inteiro 1 na fórmula para cálculo do tempo. Isto se deve ao fato que, no algoritmo da classe `CellularAutomaton` apresentado na Figura 38, os três primeiros passos são qualificados como a execução efetiva da regra do autômato celular e precedem a operação que incrementa o tempo. Assim, na última execução do algoritmo da classe `CellularAutomaton`, os três primeiros passos serão executados, porém o último passo lançará uma exceção por atingir o limite de tempo, o que representa o término da simulação e também a diferença de uma execução entre os dois algoritmos. A execução do algoritmo apresentado na Figura 38 é efetuada a partir da classe `Simulation`, controlada pelo módulo `cas-control`.

3.2.3 Módulo `cas-control`

O módulo `cas-control` reúne recursos abstratos responsáveis por controlar o ciclo de vida de uma simulação, desde a construção dos componentes do modelo de dados, o que simplifica a utilização dos recursos do módulo `cas-core`. As três principais classes desse

módulo são: `SimulationParameter`, `SimulationBuilder` e `SimulationController`. A classe `SimulationParameter` contém os objetos criados pela interface com o usuário, responsáveis por informar os parâmetros a serem utilizados para construir a simulação. A classe `SimulationBuilder` utiliza a classe `SimulationParameter` para construir o modelo de dados a ser utilizado na simulação. O algoritmo da construção do modelo de dados executado pela classe `SimulationBuilder` é apresentado na Figura 40. Cada um dos dez passos apresentados é um método abstrato e, portanto, a implementação destes ocorre na classe que estende `SimulationBuilder` através de herança, onde também é feita a definição de especificidades dimensionais.

Figura 40 – Algoritmo de construção do modelo de dados



Fonte: elaborado pelo autor.

A classe `SimulationController` é responsável por gerenciar a *thread* que executa a regra do autômato celular. Para esta execução, `SimulationController` dispõe de três *threads* diferentes: uma para a execução da simulação completa, outra para a execução de apenas uma iteração da simulação e outra para a simulação de apenas uma célula. Todas atuam sobre a mesma simulação, porém, a execução das mesmas é mutuamente exclusiva. O controle da exclusividade da execução é feito através do atributo `active` da classe `Simulation` (Figura 31). Para que uma *thread* inicie a execução da simulação, o atributo `active` precisa ser falso. O valor do atributo é alterado para verdadeiro assim que uma *thread* iniciar a execução da simulação, retornando para falso quando a *thread* concluir o processamento.

3.2.4 Módulo `cas-unidimensional`

O módulo `cas-unidimensional` é uma implementação dos módulos `cas-core` e `cas-control` que complementa as estruturas abstratas com as especificidades dos autômatos celulares unidimensionais. Cada uma das classes abstratas dos módulos `cas-core` e `cas-control` possuem pelo menos uma classe correspondente no módulo `cas-unidimensional`. As classes do módulo `cas-unidimensional` mantêm a nomenclatura das classes abstratas implementadas, porém com a inclusão do prefixo `Unidimensional`, como em `UnidimensionalSpace`, `UnidimensionalTime` e `UnidimensionalSimulationController`, entre outras.

A classe `UnidimensionalSpace` implementa os métodos abstratos da classe `Space`, conforme previsto no Quadro 2. A classe `UnidimensionalSimulationParameter` implementa a validação das informações providas da interface com o usuário, a fim de bloquear qualquer inconsistência localizada para a simulação de um autômato celular unidimensional.

A classe `UnidimensionalSimulationBuilder` implementa os 10 passos do algoritmo de construção do modelo de dados, mostrado na Figura 40. Para a construção do modelo de dados do autômato celular unidimensional elementar, são utilizadas duas instâncias da classe `UnidimensionalState`, que representam a cor branca e a cor preta. Estas duas instâncias estarão presentes em todas as transições da regra e em todas as possíveis células do espaço simuladas, o que faz de cada célula um objeto em memória com uma referência para um dos dois estados possíveis, otimizando o consumo de memória.

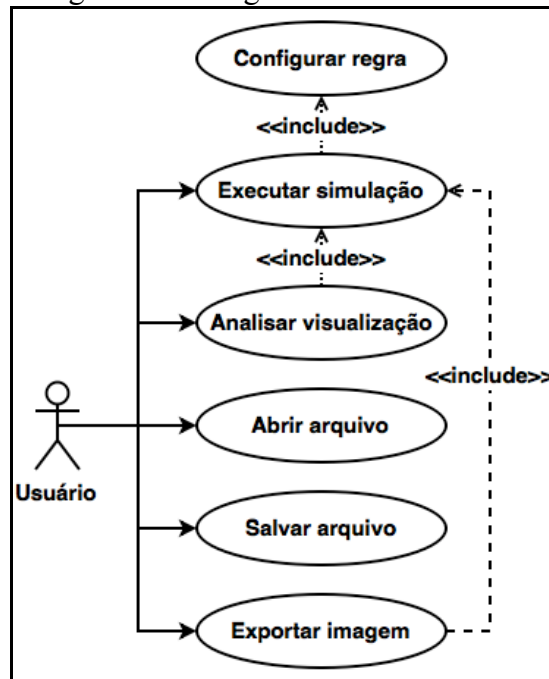
Assim como o módulo `cas-unidimensional` estende os módulos `cas-core` e `cas-control` a fim de especificar as diretivas de simulação para autômatos celulares unidimensionais, novos módulos como este podem ser desenvolvidos seguindo o mesmo princípio, para especificar as diretivas de simulação de autômatos celulares com um número de dimensões diferentes, tais como `cas-bidimensional` ou `cas-tridimensional`. Assim, o projeto pode ser estendido de forma organizada e estruturada, separando características comuns de características específicas.

3.2.5 Módulo `cas-ui-desktop`

O módulo `cas-ui-desktop` é responsável por criar a interface utilizada pelo usuário, assim como renderizar a representação da simulação dos autômatos celulares. A interface permite o usuário informar os parâmetros a serem utilizados para criar o modelo de dados da

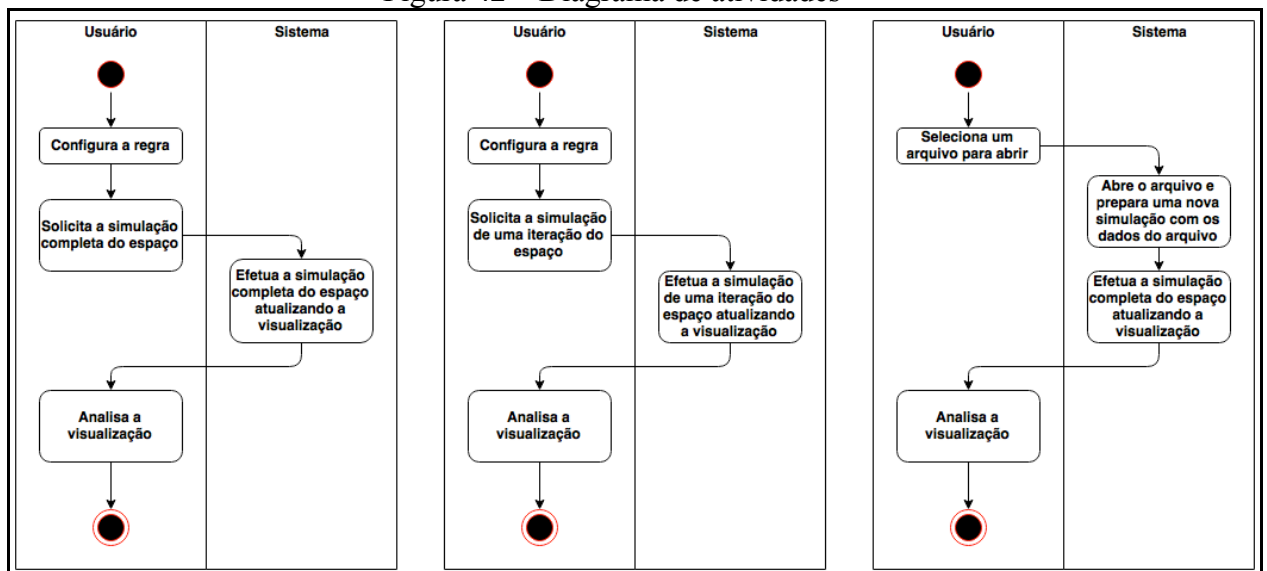
simulação, controlar a execução simulação, exportar ou salvar arquivos. A Figura 41 apresenta o diagrama de casos de uso com as operações que podem ser efetuadas pelo usuário. A Figura 42 apresenta diagramas de atividades com três das operações apresentadas no diagrama de casos de uso. A Figura 43 apresenta o diagrama de estados utilizados pela interface.

Figura 41 – Diagrama de casos de uso



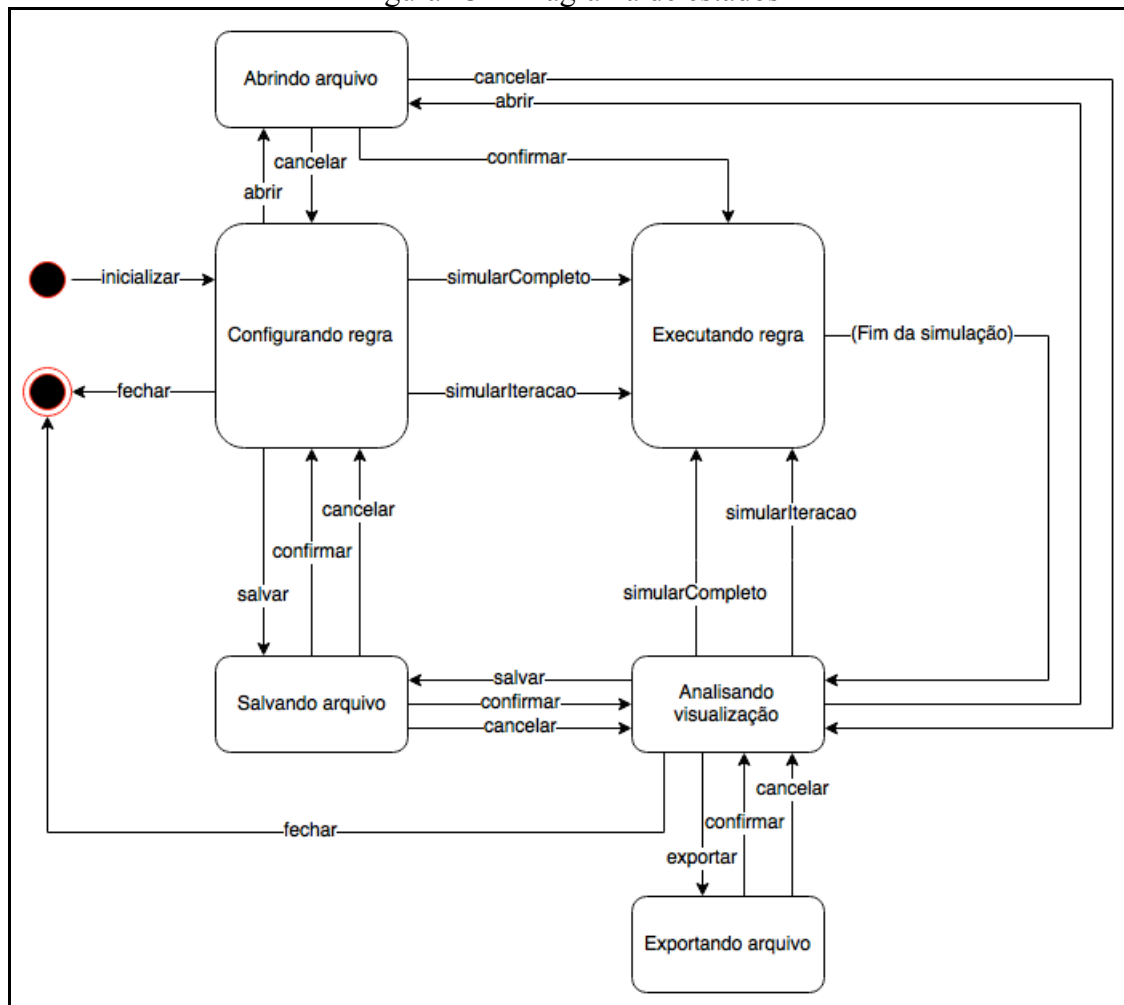
Fonte: elaborado pelo autor.

Figura 42 – Diagrama de atividades



Fonte: elaborado pelo autor.

Figura 43 – Diagrama de estados



Fonte: elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento do projeto foi adotada a plataforma de desenvolvimento Eclipse, sendo o projeto desenvolvido utilizando a linguagem Java. A estrutura do projeto através da separação por módulos e controle de dependências foi efetuada através do gerenciador de projetos Maven. O projeto foi criado sob a licença de software livre GNU Affero General Public License, Versão 3.0 (AGPLv3).

Os testes criados para os módulos utilizam a biblioteca JUnit, sendo que o módulo `cas-core` utiliza também a biblioteca Mockito, devido a sua alta abstração e necessidade de simulação de algumas estruturas durante a execução dos testes.

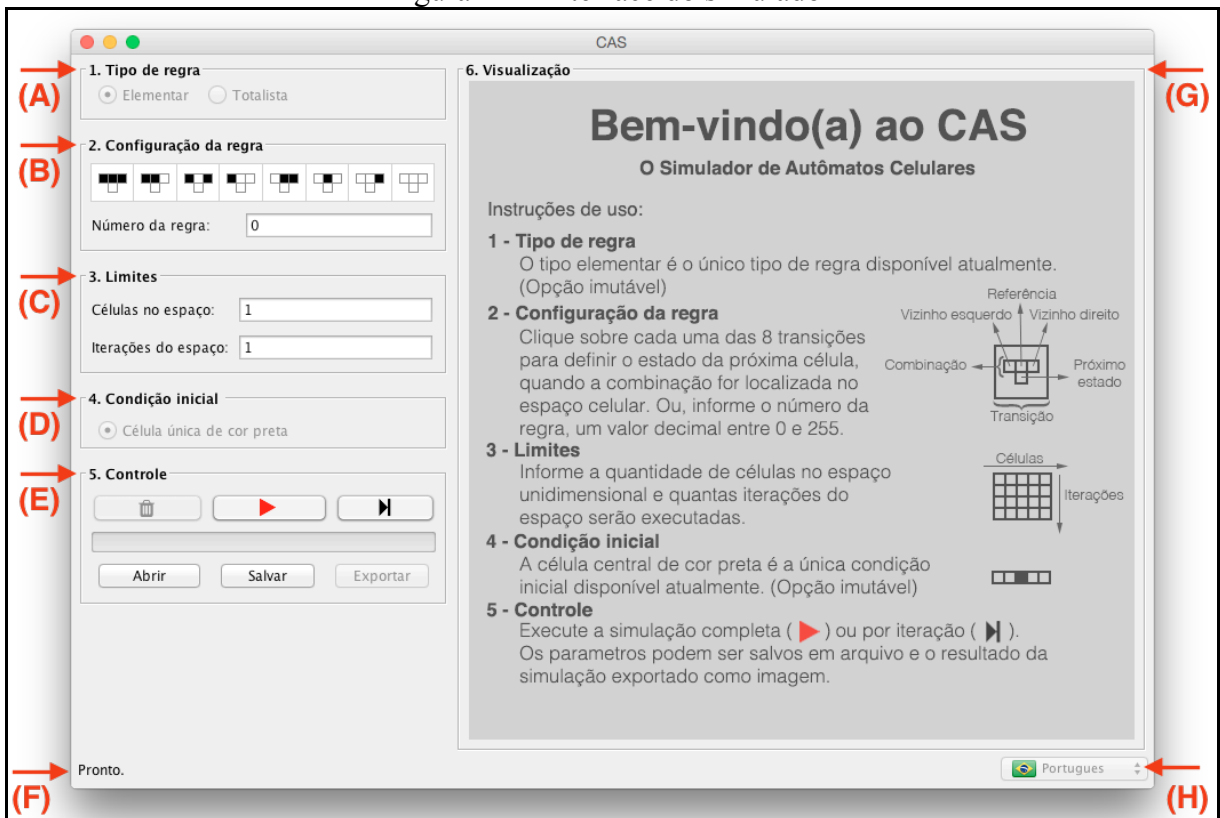
O módulo `cas-ui-desktop`, que contém a implementação da interface com o usuário, utiliza os recursos Swing e AWT disponíveis em Java, para a criação da janela principal do programa. Além destes recursos, o módulo também utiliza a biblioteca Processing, fundamental para a renderização da simulação em tempo real e criação de recursos visuais diversos. A interface possui duas *applets* Processing, uma para a configuração da regra e outra para a visualização da simulação. A biblioteca Gson foi utilizada para as operações de serialização e desserialização do arquivo que contém as propriedades de simulação, através das operações de salvar e abrir dispostas na interface.

Durante o desenvolvimento, foi utilizado o sistema de controle de versões GIT, através do serviço online GitHub, onde o repositório do projeto permanece como privado, sendo este mesmo repositório utilizado para a distribuição pública como *open source* após a conclusão do projeto. Após o desenvolvimento, o software BeyondCompare foi utilizado para efetuar a validação visual das simulações efetuadas.

3.3.2 Operacionalidade da implementação

A Figura 44 mostra a interface do simulador cuja usabilidade é orientada a uma sequência de etapas necessárias para configurar os parâmetros utilizados na simulação. Na parte esquerda da interface estão localizados os parâmetros que deve ser configurados para então executar a simulação de um autômato celular unidimensional. A primeira etapa é a configuração do tipo de regra (Figura 44A), que possui valor padrão definido como `Elementar`. O tipo da regra é imutável, pois `Totalista`, que refere-se à outra opção visível, ainda não foi implementado.

Figura 44 – Interface do simulador



Fonte: elaborado pelo autor.

A segunda etapa é a configuração da regra (Figura 44B), que possui as 8 transições possíveis para as regras elementares e também o número da regra elementar correspondente. Ao clicar sobre cada uma das 8 transições, o usuário pode definir o próximo estado da transição e o número da regra será atualizado automaticamente no campo numérico correspondente. O usuário pode, se preferir, informar um número inteiro de 0 a 255 no campo numérico, sendo as 8 transições atualizadas automaticamente de acordo com o número de regra informado.

A terceira etapa é a configuração dos limites (Figura 44C), onde deve ser informada a quantidade de células e a quantidade de iterações do espaço. Estes campos devem ser preenchidos com um número inteiro maior ou igual a 1.

A quarta etapa é a configuração da condição inicial (Figura 44D), que possui valor padrão definido como *Célula única de cor preta*. Esta etapa de configuração é imutável pois os recursos de configuração de condição inicial não foram implementados na interface. O valor padrão fará com que a condição inicial contenha apenas uma única célula de cor preta localizada no centro do intervalo de células informadas, sendo as demais células de cor branca. Isto é, se na terceira etapa de configuração (Figura 44C) for informada uma quantidade de células no espaço igual a 7, a condição inicial terá as células de 1 a 3 (da

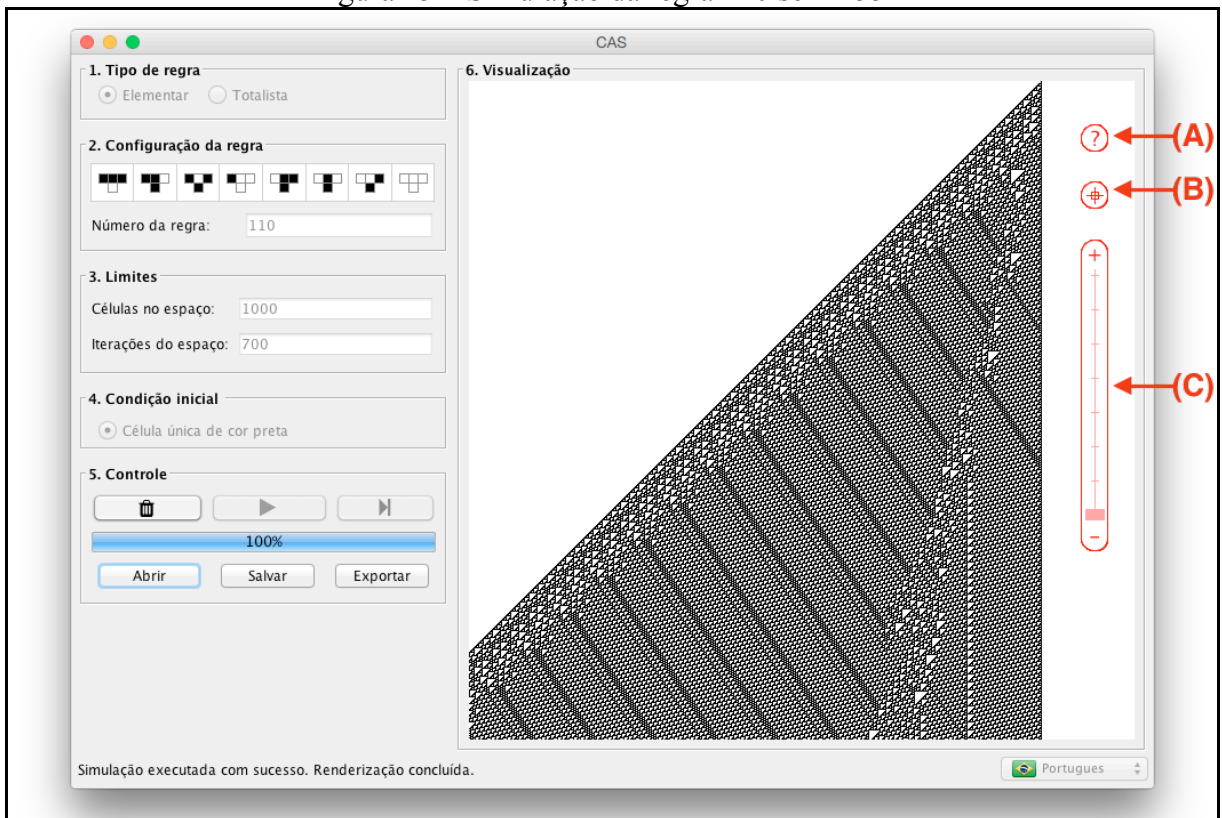
esquerda para a direita) com a cor branca, a célula 4 com a cor preta e as células 5 a 7 com a cor branca.

A área de controle (Figura 44E) possui os recursos para controle da simulação e operações de arquivo. O botão com ícone de lixeira descarta a simulação concluída ou em progresso, preparando o simulador para uma nova simulação. O botão ícone triangular vermelho (play) inicia o processo de simulação completa utilizando os parâmetros informados, sendo a simulação finalizada ao concluir a última iteração do espaço. O botão ícone triangular preto seguido de uma linha vertical (próximo) inicia o processo de simulação por iteração, onde a cada *click* é simulada e apresentada graficamente uma nova iteração do espaço, até que a última iteração seja simulada, concluindo a operação. A barra de progresso localizada entre os botões superiores e inferiores mostra o progresso da renderização da simulação em tempo real. O botão `Abrir` permite carregar um arquivo com parâmetros de simulação salvo anteriormente. O botão `Salvar` permite salvar um arquivo com os parâmetros da simulação atual. O botão `Exportar` permite exportar um arquivo em formato PNG com a representação visual da simulação, onde cada célula será representada por 1 pixel.

A barra de status apresentada na Figura 44 (F) mostra mensagens de acordo com o estado do simulador. As mensagens possíveis são: mensagem de erro ao informar um parâmetro de configuração inválido; mensagem caso ocorra algum erro inesperado durante a simulação, e mensagens de sucesso ao concluir a simulação ou operações em arquivo. A opção de linguagem (Figura 44H) possui o valor padrão definido como `Português`. Esta opção é imutável, pois outras opções de linguagem ainda não foram implementadas.

A área de visualização mostrada na Figura 44 (G) apresenta inicialmente uma mensagem de boas vindas, que contém instruções de uso (à esquerda) e alguns pictogramas legendados (à direita) para facilitar o uso do simulador. Assim que o usuário inicia a simulação, a área de visualização é alterada mostrando o resultado da simulação e alguns recursos didáticos de análise. A Figura 45 mostra a simulação completa de 700 iterações da regra elementar 110, a partir de uma condição inicial que contém 1000 células.

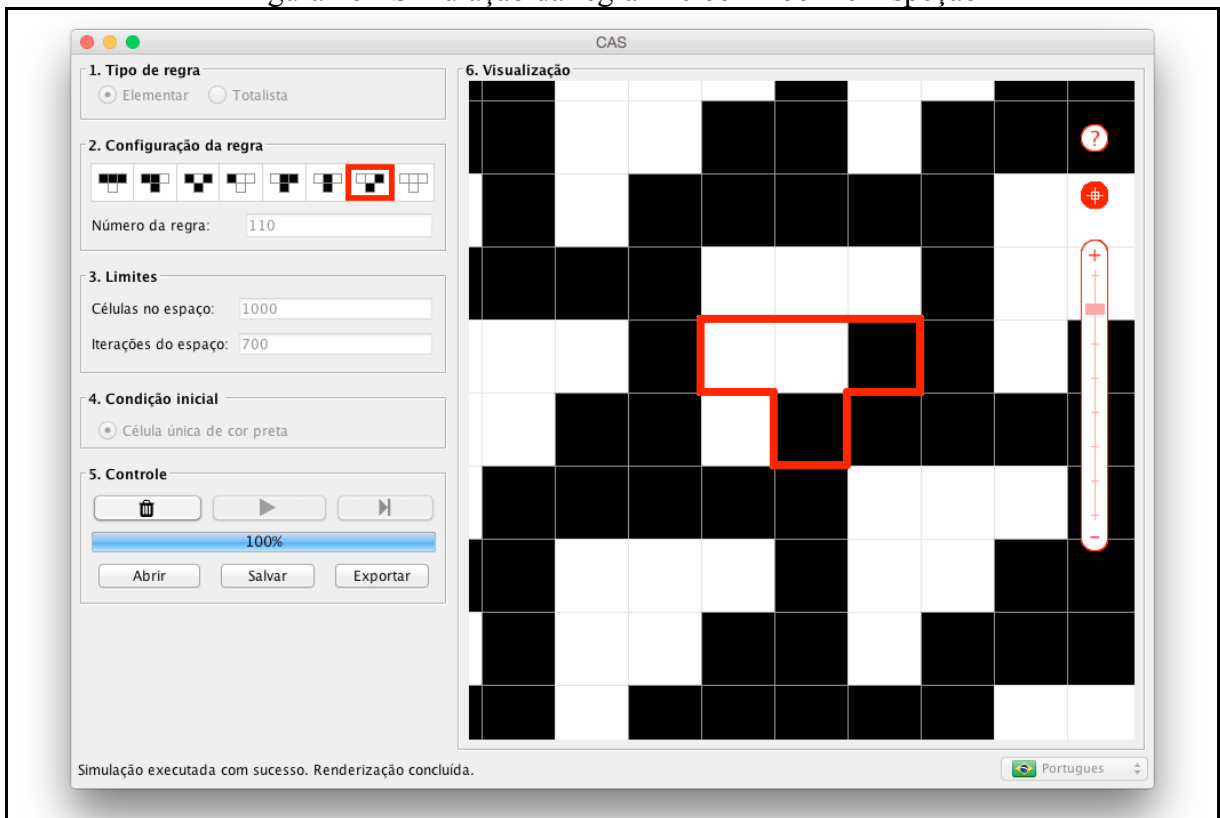
Figura 45 – Simulação da regra 110 sem zoom



Fonte: elaborado pelo autor.

Como pode ser visto na Figura 45, a tela de boas vindas é substituída pela visualização da simulação assim que é iniciado o processo e simulação. Também na área de visualização, são exibidos os recursos de ajuda (A), inspeção (B) e zoom (C). Ao clicar sobre o botão de ajuda (A), é exibida uma janela com as mesmas instruções da janela de boas vindas. O botão de inspeção de células (B) permite revelar a transição que deu origem a uma célula qualquer da área de visualização. Clicando sobre esse botão, o ponteiro do mouse será alterado para uma cruz, permitindo clicar em qualquer célula. Desse modo, ao clicar em uma célula, o simulador criará um contorno vermelho intermitente ao redor da célula escolhida e de sua vizinhança na iteração anterior, apresentando também, um contorno na transição utilizada entre as duas iterações envolvidas. O inspetor pode ser desabilitado com a tecla ESC ou com um novo clique sobre o botão de inspeção (B). A barra de zoom (C) pode ser utilizada para visualizar as linhas de contorno do espaço celular e também para melhorar a visualização da área inspecionada. A Figura 46 apresenta a mesma simulação da regra 110 utilizando zoom e o recurso de inspeção.

Figura 46 - Simulação da regra 110 com zoom e inspeção

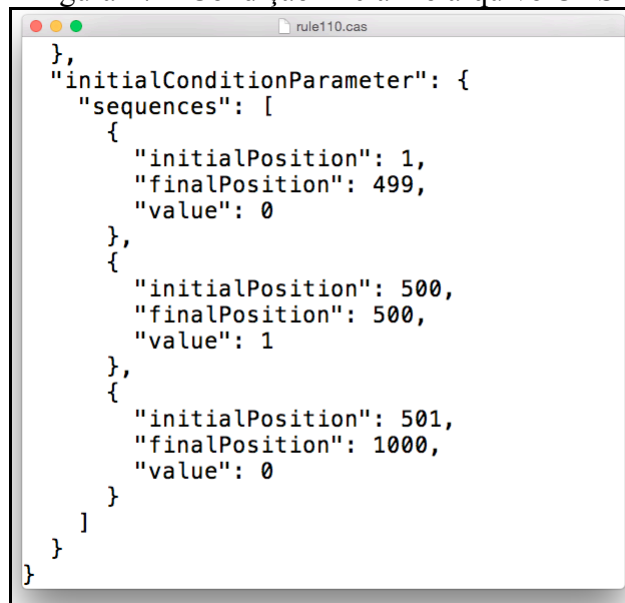


Fonte: elaborado pelo autor.

Além dos recursos já apresentados, toda a área de visualização da simulação permite o deslocamento do espaço celular para qualquer direção. Para mover o espaço celular, basta clicar sobre a área de visualização e mover o mouse para a área desejada, ainda com o botão do mouse pressionado. Ao liberar o botão do mouse, a área do espaço celular será movida de forma correspondente.

Apesar da interface não permitir a definição de condições iniciais além da célula central de cor escura, é possível definir condições iniciais diferentes utilizando o arquivo salvo pelo próprio simulador. O arquivo de extensão `.cas`, criado pelo simulador quando o usuário salva os parâmetros de simulação, é um arquivo texto em formato JSON. Neste arquivo, o elemento `initialConditionParameter` representa uma lista de sequências de células da mesma cor, conforme pode ser observado na Figura 47, onde são visualizadas as três sequências responsáveis por formar a condição inicial padrão, para um autômato celular unidimensional com 1000 células no espaço. Cada sequência tem o número da primeira célula (elemento `initialPosition`), o número da célula da última célula (elemento `finalPosition`) e a cor utilizada pelas células desta sequência (elemento `value`), onde 0 representa a cor branca e 1 representa a cor preta.

Figura 47 – Condição inicial no arquivo CAS



```

},
"initialConditionParameter": {
  "sequences": [
    {
      "initialPosition": 1,
      "finalPosition": 499,
      "value": 0
    },
    {
      "initialPosition": 500,
      "finalPosition": 500,
      "value": 1
    },
    {
      "initialPosition": 501,
      "finalPosition": 1000,
      "value": 0
    }
  ]
}
}

```

Fonte: elaborado pelo autor.

3.4 ANÁLISE DOS RESULTADOS

Para a validação do simulador, foram escolhidas três regras elementares – regra 30, regra 90 e regra 110 – utilizadas em simulações no projeto desenvolvido e no trabalho correlato Mathematica. Os resultados obtidos foram comparados em três avaliações diferentes: lógica, visual e de tempo, apresentadas nas próximas seções.

3.4.1 Avaliação lógica

O objetivo da avaliação lógica é garantir que a simulação efetuada pelo simulador desenvolvido possui equivalência total à simulação efetuada pelo trabalho correlato Mathematica. A fim de estabelecer o modelo utilizado para a comparação com o simulador desenvolvido, foram exportados três arquivos do Mathematica, cada um contendo o resultado da simulação de uma das três regras selecionadas para a comparação. Cada arquivo foi utilizado por um teste JUnit que executa a simulação de uma regra no simulador desenvolvido, comparando o resultado obtido com o conteúdo do arquivo exportado do Mathematica.

Os arquivos das regras 30 e 90 possuem o resultado de 1000 iterações de uma condição inicial de 1000 células. A condição inicial possui uma célula central de cor preta, ou seja, as células 1 até 499 são da cor branca, a célula 500 possui cor preta e as células 501 até 1000 são da cor branca. O arquivo da regra 110 também possui 1000 iterações de uma condição inicial de 1000 células, porém em sua condição inicial, a célula de cor preta é a última à direita, sendo as células 1 até 999 da cor branca e a célula 1000 da cor preta. A condição inicial da

regra 110 é diferente da condição inicial das demais regras devido ao fato de que esta regra não utiliza o espaço celular à direita, evoluindo apenas para a esquerda. Observa-se que os arquivos exportados do Mathematica contêm 1001 linhas, que correspondem à condição inicial e suas 1000 iterações, sendo que cada uma das linhas contém 1000 números inteiros representando o estado de cada uma das células, onde 0 representa a cor branca e 1 representa a cor preta.

Cada um dos testes JUnit efetua a simulação de uma regra e mantém o resultado em memória. Em seguida, o arquivo exportado pelo Mathematica com a simulação da regra correspondente é carregado, iniciando a comparação de todos os valores inteiros de forma sequencial. Para cada uma das regras são efetuadas 1.001.000 operações de comparação entre uma célula simulada através do trabalho proposto com a mesma célula simulada através do Mathematica, sendo o resultado do teste positivo apenas se o estado das 1.001.000 células de cada simulação forem equivalentes. A Figura 48 mostra o resultado de uma execução dos três testes JUnit, onde cada um dos testes validou com sucesso a simulação de cada regra em 1,698 segundos (s).

Figura 48 – Validação lógica das regras elementares 30, 90 e 110



Fonte: elaborado pelo autor.

3.4.2 Avaliação visual

A avaliação visual é uma validação redundante do processo de avaliação lógica. Esta avaliação compara a representação gráfica das três regras elementares a fim de garantir que suas visualizações também são equivalentes. Para a validação visual, foram gerados três arquivos de imagem em formato PNG através do Mathematica, onde cada arquivo contém a representação visual da simulação de cada regra.

Em todos os arquivos, cada pixel representa uma célula, sendo seus estados representados através das cores branca e preta. A condição inicial, quantidade de células e quantidade de iterações utilizadas para a exportação dos arquivos de imagem são as mesmas utilizadas na exportação dos arquivos de texto, sob a qual foi efetuada a avaliação lógica.

Desta forma, as dimensões dos arquivos de imagem são de 1.000 pixels horizontais e 1001 pixels verticais.

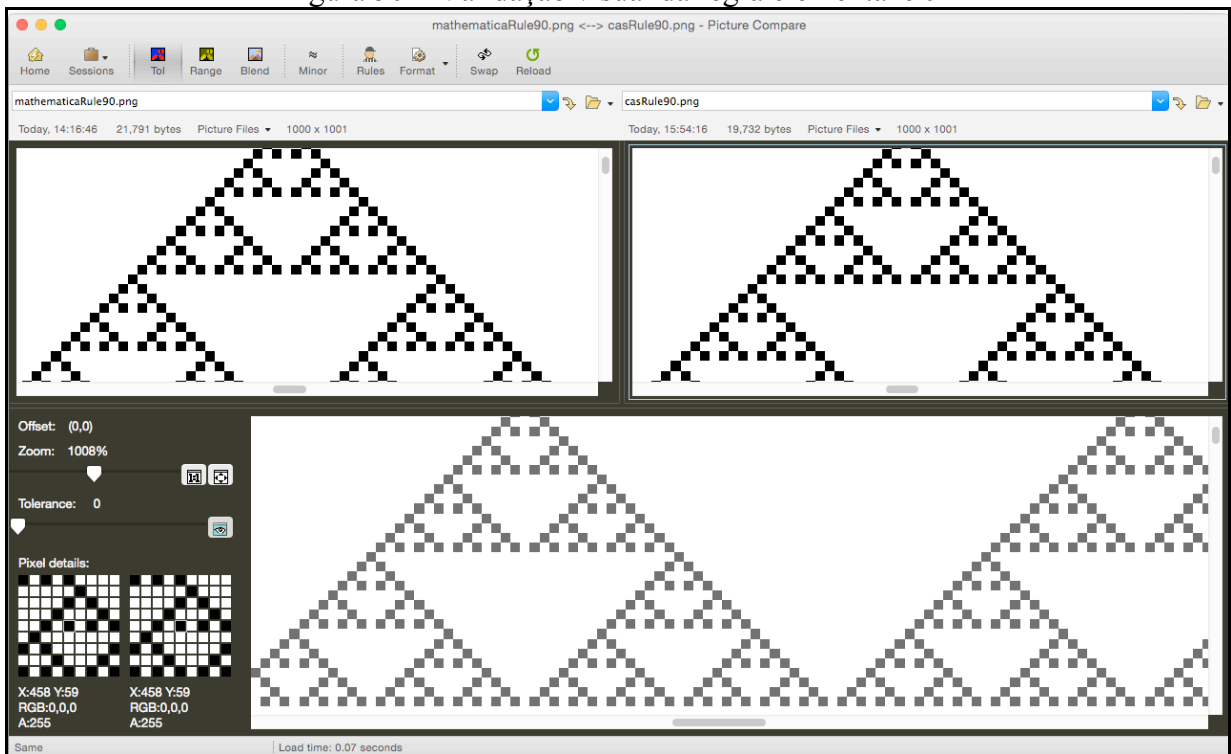
Para gerar as imagens em formato PNG no simulador desenvolvido, foi utilizado o botão `Exportar` após a simulação de cada uma das regras. Os arquivos de imagem foram comparados utilizando o software `BeyondCompare` que, após avaliar todos os pixels, informou o resultado de equivalência entre todas as imagens. A Figura 49 mostra a comparação da regra 30, a Figura 50 mostra a comparação da regra 90 e a Figura 51 mostra a comparação da regra 110. Nas figuras, o padrão superior à esquerda é do arquivo exportado pelo `Mathematica`, o padrão superior à direita é do arquivo exportado pelo simulador desenvolvido e o padrão inferior é a visualização de diferenças, as quais, se existirem, são destacada em vermelho. Os padrões visualizados nestas figuras utilizam aproximadamente 1000% de zoom a partir da imagem original e o resultado da comparação é mostrado na barra de status do software `BeyondCompare`, apresentando *Same* (idêntico) em todas as comparações.

Figura 49 – Validação visual da regra elementar 30



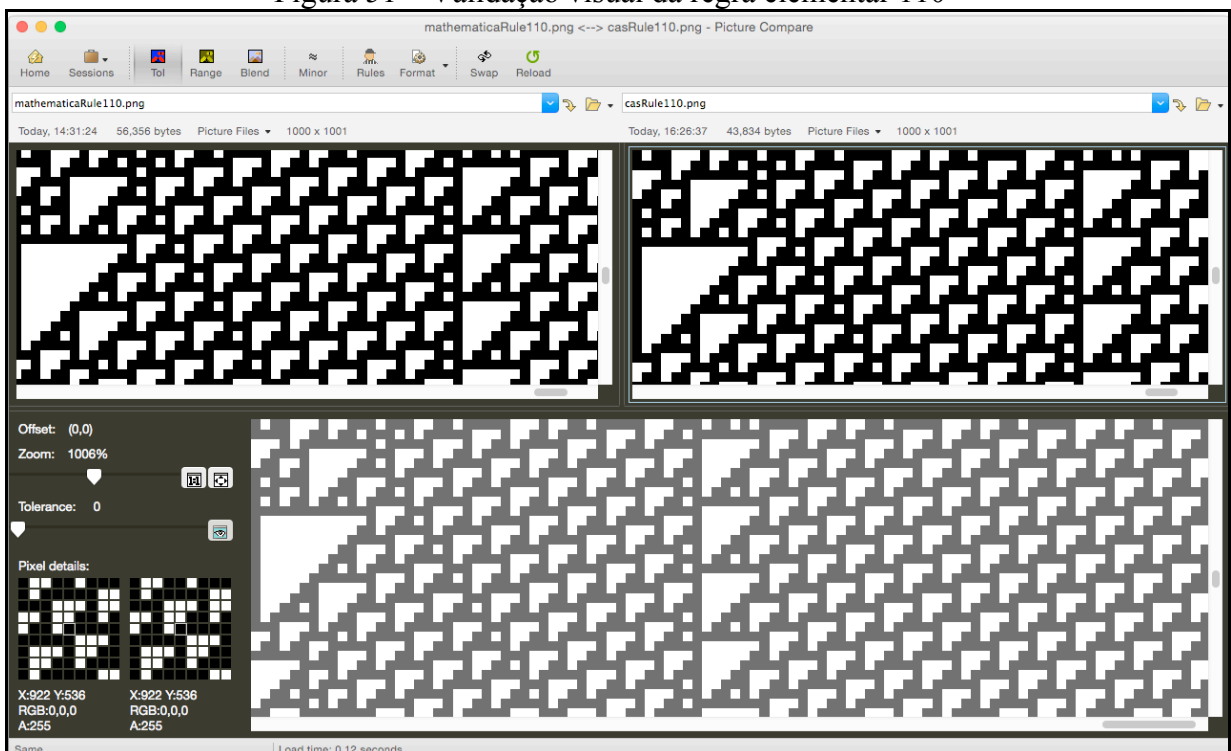
Fonte: elaborado pelo autor.

Figura 50 – Validação visual da regra elementar 90



Fonte: elaborado pelo autor.

Figura 51 – Validação visual da regra elementar 110



Fonte: elaborado pelo autor.

3.4.3 Avaliação de tempo

A avaliação de tempo consiste na comparação dos tempos utilizados pelo simulador desenvolvido e pela ferramenta Mathematica, para efetuar a computação da mesma regra

elementar, utilizando os mesmos parâmetros. Para esta avaliação, apenas a regra 110 foi utilizada. Para a coleta de dados foi estabelecido que, devido as possíveis variações de tempo entre simulações, o tempo da simulação considerado é uma média dos tempos de 10 simulações diferentes em cada ferramenta, trazendo maior consistência ao resultado final. A condição inicial, quantidade de células e quantidade de iterações utilizadas para simular a regra 110 nas duas ferramentas são as mesmas utilizadas nas avaliações visual e lógica. O tempo de execução do Mathematica foi obtido através da execução do comando `AbsoluteTiming`, já do simulador desenvolvido foi utilizado o tempo de execução da simulação através do JUnit.

A Tabela 1 apresenta os tempos, em segundos, de 10 simulações diferentes no Mathematica e no simulador desenvolvido, sendo a última linha a média entre os tempos de cada ferramenta. Os testes foram efetuados utilizando um computador com processador Intel Core i5 com 2.3GHz de velocidade e 2 núcleos, 16GB de memória RAM DDR3 com 1333MHz de velocidade e unidade de armazenamento de estado sólido com 960GB e 6Gbps de velocidade.

Tabela 1 – Consumo de tempo para simulação da regra 110

#	Mathematica	CAS
1	0.005533	0.392
2	0.005704	0.359
3	0.00574	0.359
4	0.005623	0.382
5	0.005584	0.380
6	0.005571	0.358
7	0.005552	0.353
8	0.006483	0.359
9	0.005373	0.362
10	0.00533	0.359
Média:	0.0056493	0.3663

Fonte: elaborado pelo autor.

Conforme os dados apresentados, o Mathematica é aproximadamente 65 vezes mais rápido do que o simulador desenvolvido. Esta diferença pode estar associada a diversos fatores, desde as tecnologias utilizadas para a construção de cada software até a técnica adotada para executar a simulação. Como o Mathematica é um recurso de código protegido, os detalhes de nível estrutural são desconhecidos. Porém, considerando a sofisticação e robustez da plataforma, existe a possibilidade de que o seu processamento ocorra de forma paralela, processando múltiplas células em uma mesma iteração. O simulador construído opera de forma sequencial, processando apenas uma célula por vez durante as iterações.

3.4.4 Comparativo entre os trabalhos correlatos

O Quadro 3 apresenta um comparativo entre as características mais relevantes dos trabalhos correlatos e as funcionalidades desenvolvidas para o simulador CAS.

Quadro 3 – Comparativo entre os trabalhos correlatos e o simulador CAS

trabalhos relacionados características	CAS	Golly	Mathematica
suporte a espaços unidimensionais (1D), bidimensionais (2D), tridimensionais (3D)	1D	1D 2D	todos
suporte a autômatos celulares elementares (E), gerais (G), totalistas (T), totalistas exteriores (TE)	E	E T TE	todos
permite a execução passo-a-passo	X	X	X
permite a configuração da regra	X	X	X
permite a configuração do número de iterações	X		X
permite salvar a configuração de um autômato em arquivo	X	X	X
permite salvar o resultado da computação de um autômato como imagem	X		X
o propósito didático prevalece ao propósito científico	X		
permite visualizar a transição da regra utilizada na criação das células	X		

Fonte: elaborado pelo autor.

Como pode ser visualizado no Quadro 3, o simulador CAS desenvolvido é limitado em relação a quantidade de dimensões do espaço celular, assim como quanto aos tipos de autômatos celulares. Suporta apenas o espaço unidimensional e os autômatos celulares elementares, embora o seu modelo de dados favoreça a implementação dos demais tipos de autômatos em espaços com mais dimensões.

A execução passo-a-passo e a configuração da regra são características que podem ser verificadas em todas as ferramentas. O número de iterações pode ser determinado no CAS e no Mathematica, sendo que os algoritmos de Golly evoluem constantemente o espaço celular em novas gerações, permitindo que a execução seja iniciada e interrompida, porém não permite informar um número finito de gerações ou iterações desejadas.

Todas as ferramentas permitem salvar a configuração de um autômato em arquivo, porém Golly não permite salvar o resultado da computação de um autômato como imagem. O simulador CAS permite exportar o resultado da computação de um autômato no formato PNG e o Mathematica suporta tanto o formato PNG quanto outros formatos.

O propósito didático prevalece ao propósito científico apenas no CAS, onde foram criados recursos para auxiliar tanto os usuários que não conhecem o tema quanto aqueles que

conhecem, porém nunca utilizaram o simulador. A janela de boas vindas do CAS possui uma explicação geral de como o simulador deve ser utilizado. Os recursos visuais são integrados de uma forma dinâmica, permitindo uma experiência mais intuitiva do que nos trabalhos correlatos. Além disso, é possível conhecer como funcionam os autômatos celulares unidimensionais através do recurso de inspeção, que apresenta a transição da regra utilizada para gerar cada uma das células, sendo este um recurso exclusivo do CAS em relação aos trabalhos correlatos.

4 CONCLUSÕES

Trabalhar com o tema de autômatos celulares foi desafiador e gratificante. Este denso assunto possui uma diversidade de aplicações permitindo aprofundamento prático, assim como uma complexidade relevante para linhas de pesquisa.

O simulador de autômatos celulares unidimensionais desenvolvido alcançou todos os objetivos propostos com sucesso, sendo apenas um de seus requisitos funcionais atendido parcialmente ao não disponibilizar a configuração de condição inicial através da interface gráfica, porém permitindo esta configuração através de arquivo.

As ferramentas e tecnologias utilizadas ao longo do projeto atenderam com qualidade as demandas de desenvolvimento, teste, validação, experimentação, entre outras. Algumas instabilidades foram notadas durante a renderização efetuada pela biblioteca Processing, afetando as operações de translação e escala. Porém, devido ao cronograma e à baixa ocorrência de tal instabilidade, não houve investigação aprofundada.

A adoção de uma perspectiva abstrata durante o desenvolvimento do projeto teve um impacto positivo, permitindo que o simulador implemente novos recursos tais como espaços multidimensionais e tipos diferentes de autômatos celulares. Além disso, os elementos de propósito didático, embora concluídos, podem ser otimizados com um estudo maior sobre a experiência de usuário.

Com o simulador de autômatos celulares unidimensionais CAS, torna-se mais fácil a apresentação e estudo da área de autômatos celulares para pessoas que ainda não conhecem o tema, ou que desejam aprofundar o seu conhecimento. A utilização do simulador por usuários de diferentes áreas do conhecimento pode inspirar novas aplicações para o tema, ajudando ainda a sua popularização. Considerando ainda a sua distribuição como software livre, a sua estrutura e modelo de dados escritos em Java podem contribuir para análises aprofundadas sobre o conceito, tanto para sua utilização e integração com outras aplicações como para finalidade científica.

4.1 EXTENSÕES

Como sugestão de extensões para o simulador propõe-se:

- a) suporte a múltiplos idiomas;
- b) configuração da condição inicial através da interface gráfica, incluindo opção de condição inicial randômica;
- c) substituição da interface Swing por uma biblioteca de interface Processing como ControlP5;

- d) suporte à redimensionamento da janela e modo tela-cheia;
- e) novos formatos de exportação de arquivo, principalmente um formato em texto plano;
- f) implementação de outros tipos de regras em espaço unidimensional, tais como: geral, totalista e totalista exterior;
- g) implementação bidimensional do módulo `cas-core` e criação dos recursos de interface correspondentes;
- h) implementação de tipos de regras em espaço bidimensional, tais como: geral, totalista e totalista exterior;
- i) implementação tridimensional do módulo `cas-core` e criação dos recursos de interface correspondentes;
- j) implementação de tipos de regras em espaço tridimensional, tais como: geral, totalista e totalista exterior;
- k) suporte à configuração de vizinhanças de acordo com a quantidade de dimensões do espaço;
- l) suporte ao processamento paralelo.

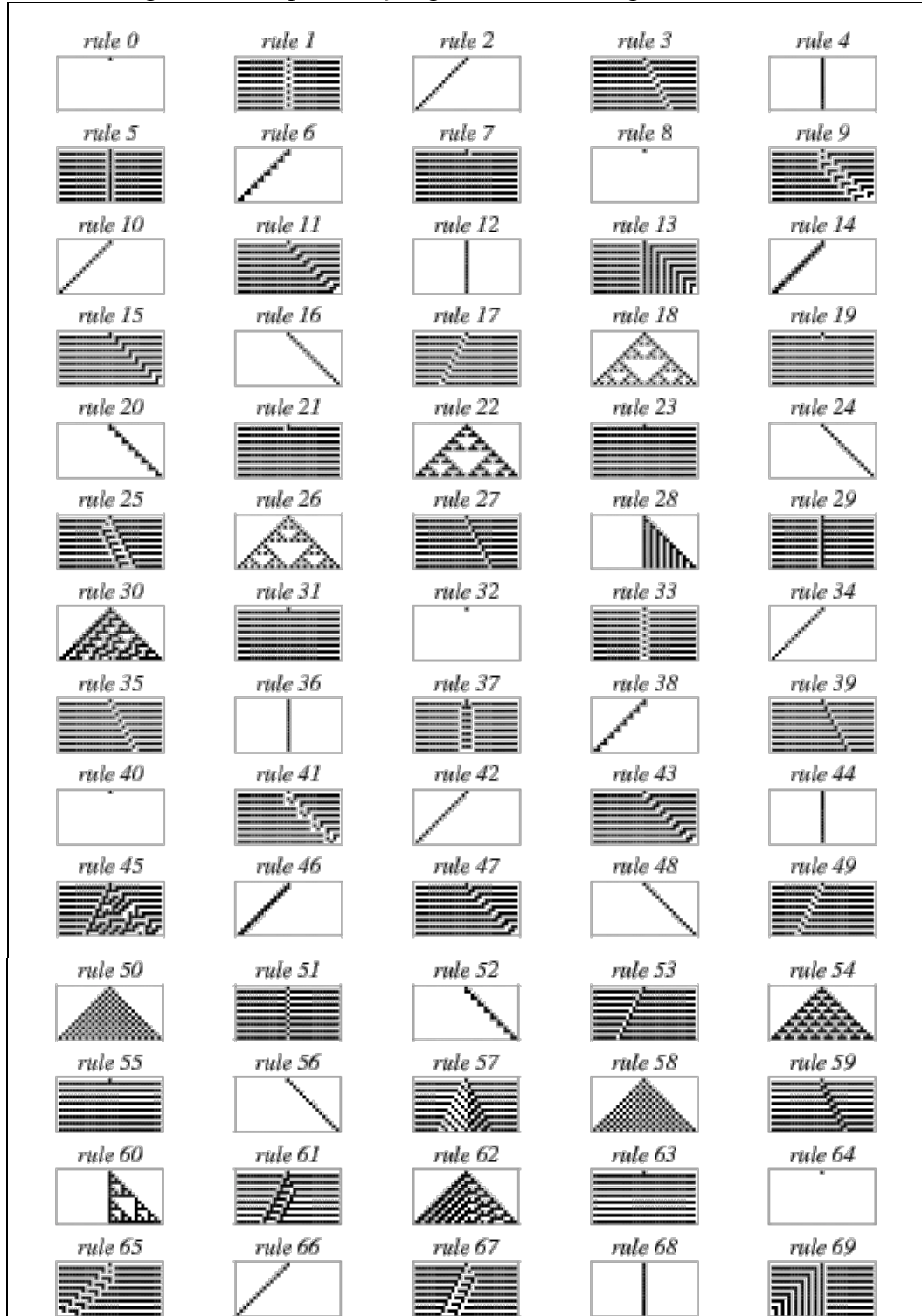
REFERÊNCIAS

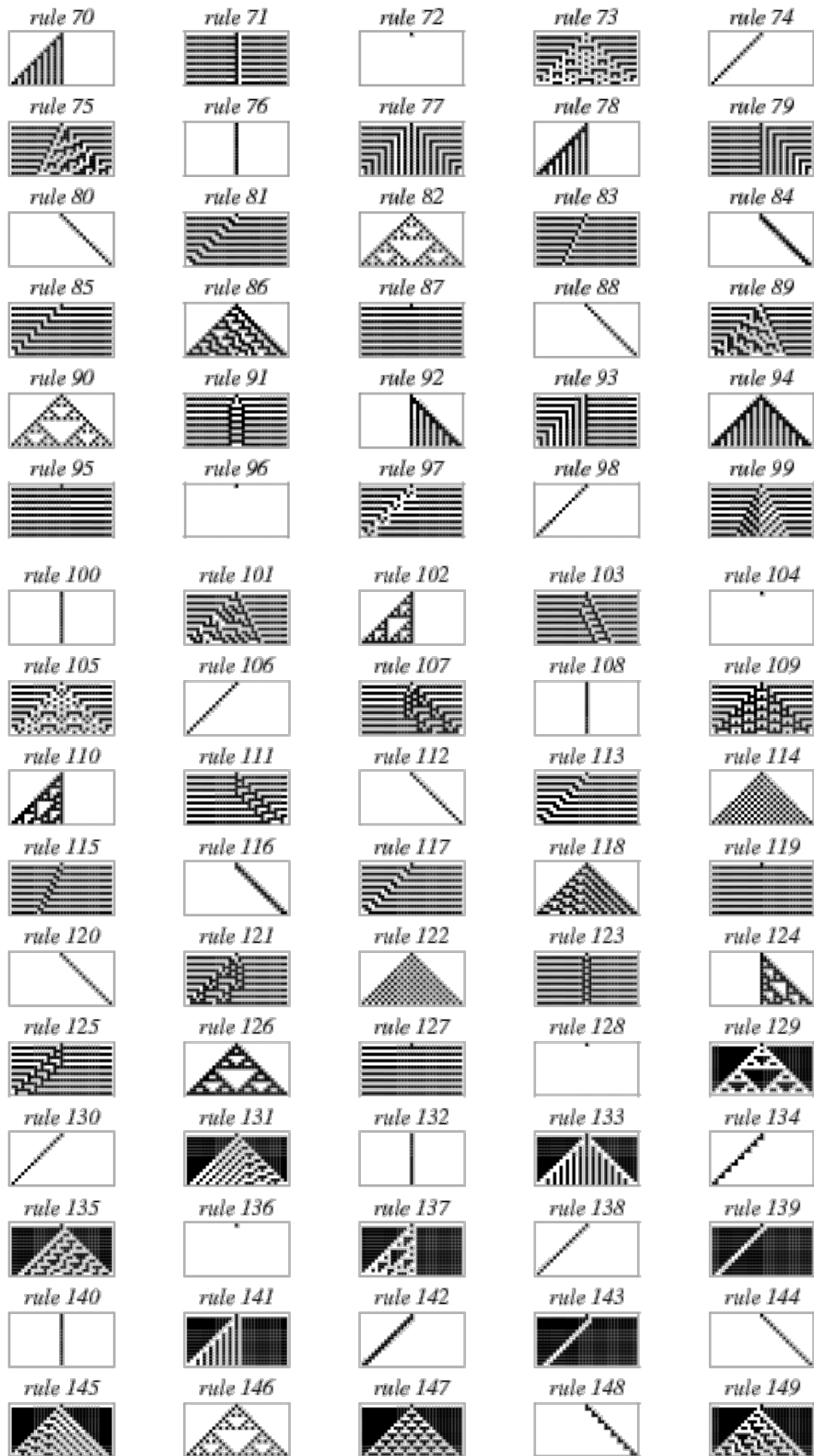
- BANKS, Jerry et al. **Discrete-event system simulation**. 4th ed. Harlow: Pearson Education, 2005.
- COOCK, Matthew. **Universality in elementary cellular automata**. [S.l.], 2004. Disponível em: <<http://www.complex-systems.com/pdf/15-1-1.pdf>>. Acesso em: 12 nov. 2016.
- _____. **A concrete view of rule 110 computation**. [S.l.], 2008. Disponível em: <<https://arxiv.org/pdf/0906.3248v1.pdf>>. Acesso em: 12 nov. 2016.
- FLOREANO, Dario; MATTIUSI, Claudio. **Bio-inspired artificial intelligence: theories, methods, and technologies**. Cambridge: MIT Press, 2008.
- ILACHINSKI, Andrew. **Cellular automata: a discrete universe**. Singapore: World Scientific Publishing, 2001.
- MCINTOSH, Harold V. **One dimensional cellular automata**. Frome: Luniver Press, 2009.
- PICKOVER, Clifford A. **The math book: from pythagoras to the 57th dimension, 250 milestones in the history of mathematics**. New York: Sterling Publishing Co, 2009.
- ROKICKI, Tomas et al. **Golly: a game of life simulator**. [S.l.], 2015. Disponível em: <<http://golly.sourceforge.net>>. Acesso em: 02 abr. 2016.
- SCHIFF, Joel L. **Cellular automata: a discrete view of the world**. New Jersey: John Wiley & Sons, 2008.
- SMITH, Roger D. **Simulation article**. [S.l.], 1998. Disponível em: <<http://www.modelbenders.com/encyclopedia/encyclopedia.html>>. Acesso em: 15 nov. 2016.
- WEISSTEIN, Eric W. **Elementary cellular automaton**. [S.l.], 2009. Disponível em: <<http://mathworld.wolfram.com/ElementaryCellularAutomaton.html>>. Acesso em: 02 abr. 2016.
- WOLFRAM, Stephen. **A new kind of science**. Champaign: Wolfram Media, 2002.
- _____. Statistical mechanics of cellular automata. **Reviews of Modern Physics**, [S.l.], v. 55, n. 3, p. 601-644, July 1983. Disponível em: <<http://www.stephenwolfram.com/publications/academic/statistical-mechanics-cellular-automata.pdf>>. Acesso em: 03 abr. 2016.
- WOLFRAM ALPHA LLC . **Wolfram Mathematica: sistema definitivo para computação técnica moderna**. [S.l.], 2016. Disponível em: <<https://www.wolfram.com/mathematica/>>. Acesso em: 04 abr. 2016.

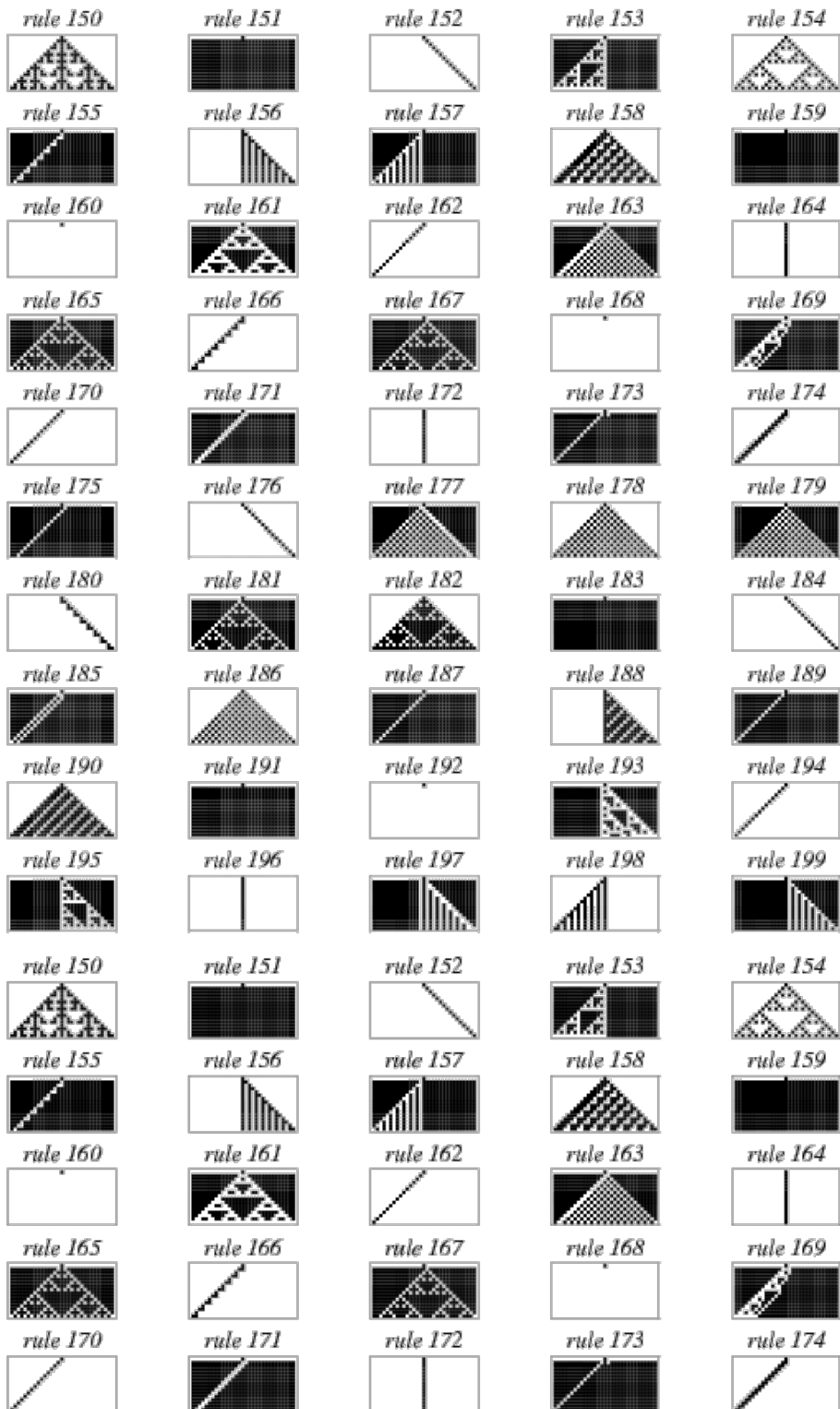
APÊNDICE A – Regras elementares

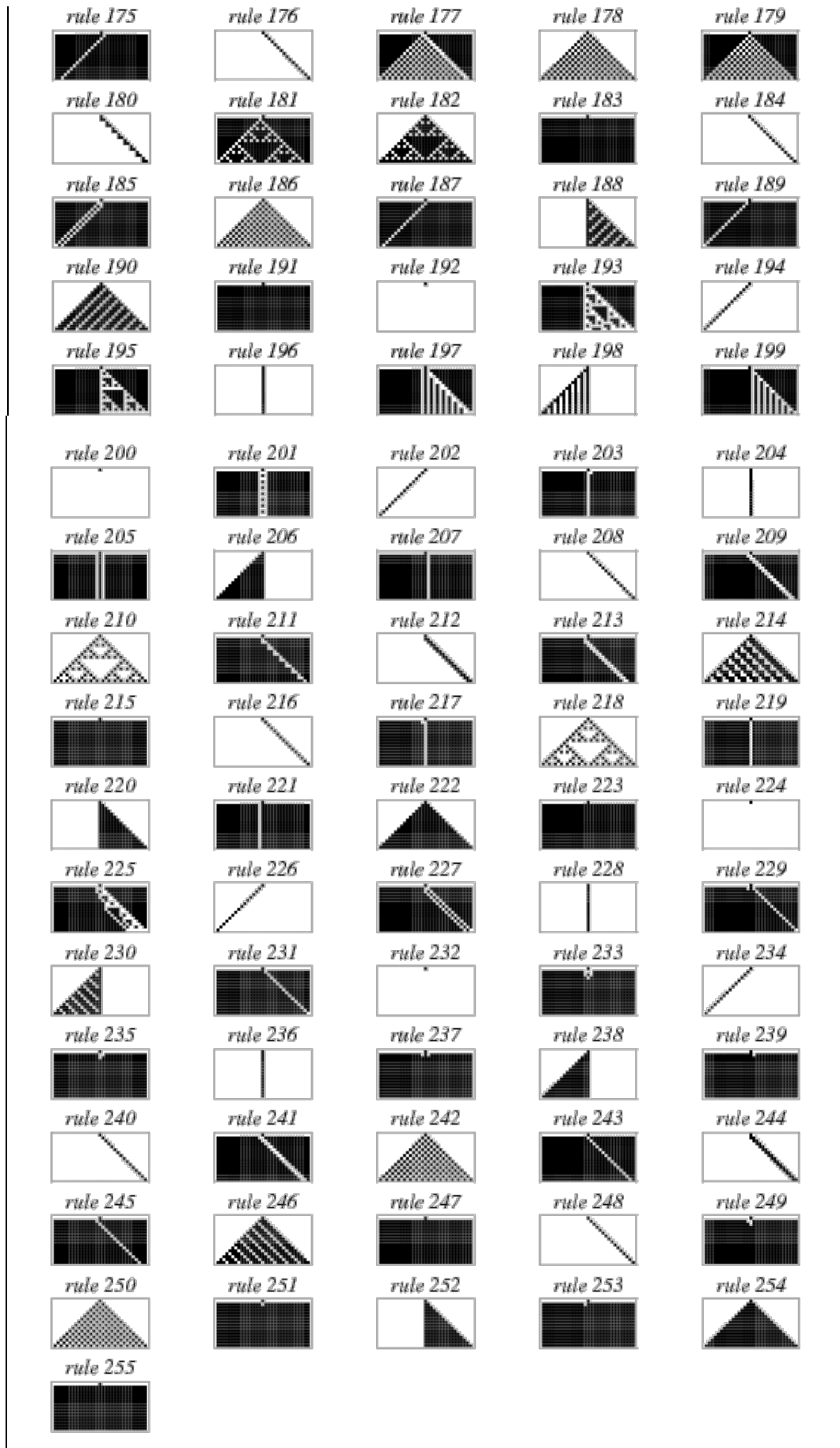
A Figura 52 representa graficamente 15 iterações de cada uma das 256 possíveis regras elementares para autômatos celulares unidimensionais, com condição inicial equivalente a uma célula central de cor preta.

Figura 52 – Representação gráfica das 256 regras elementares







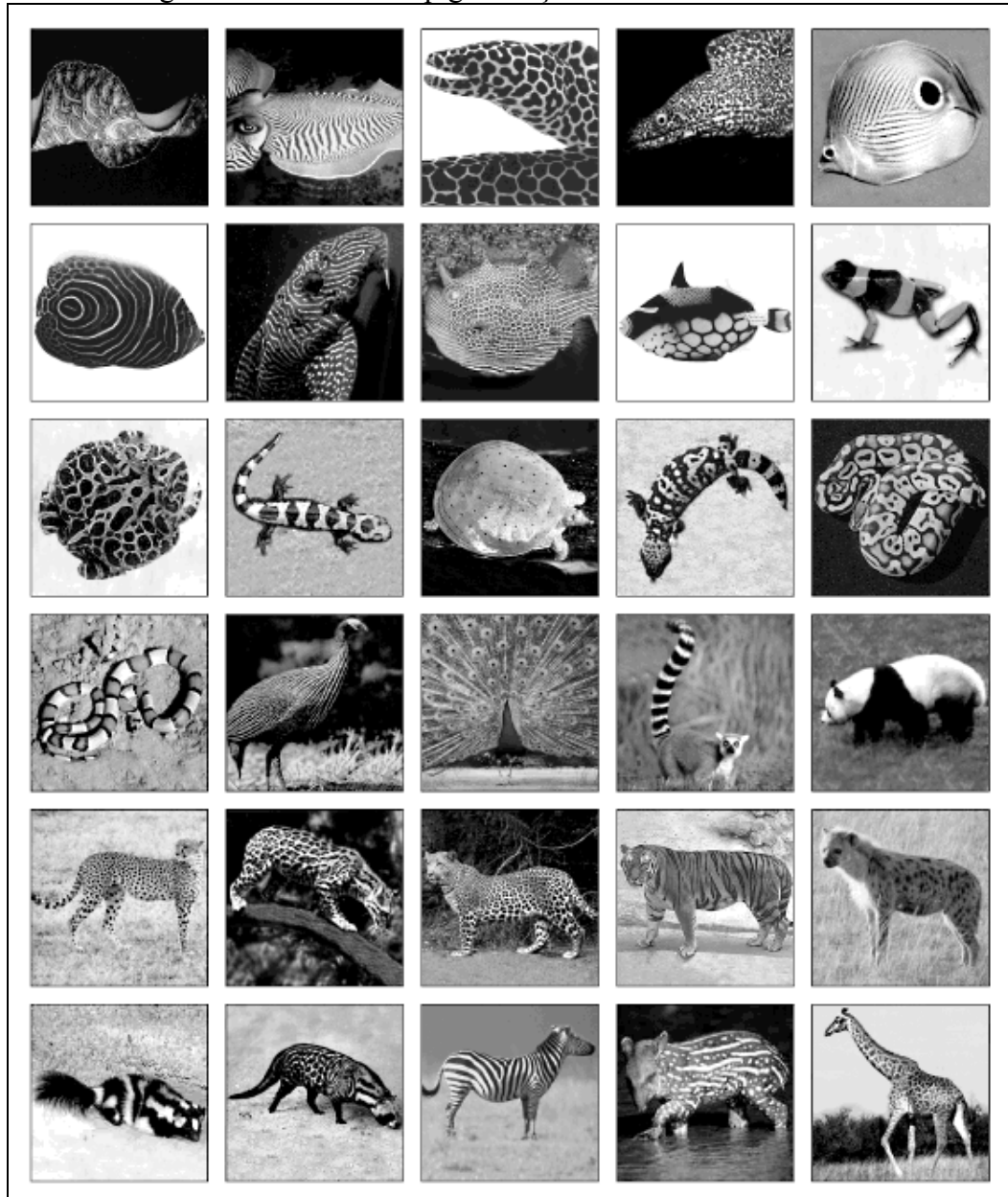


Fonte: Weisstein (2009).

APÊNDICE B – Padrões biológicos

Conforme apresentado por Wolfram (2002), a manifestação de padrões de pigmentação em meio natural possuem similaridade aos padrões reproduzidos por regras simples de autômatos celulares. A Figura 53 apresenta padrões de pigmentação encontrados em animais.

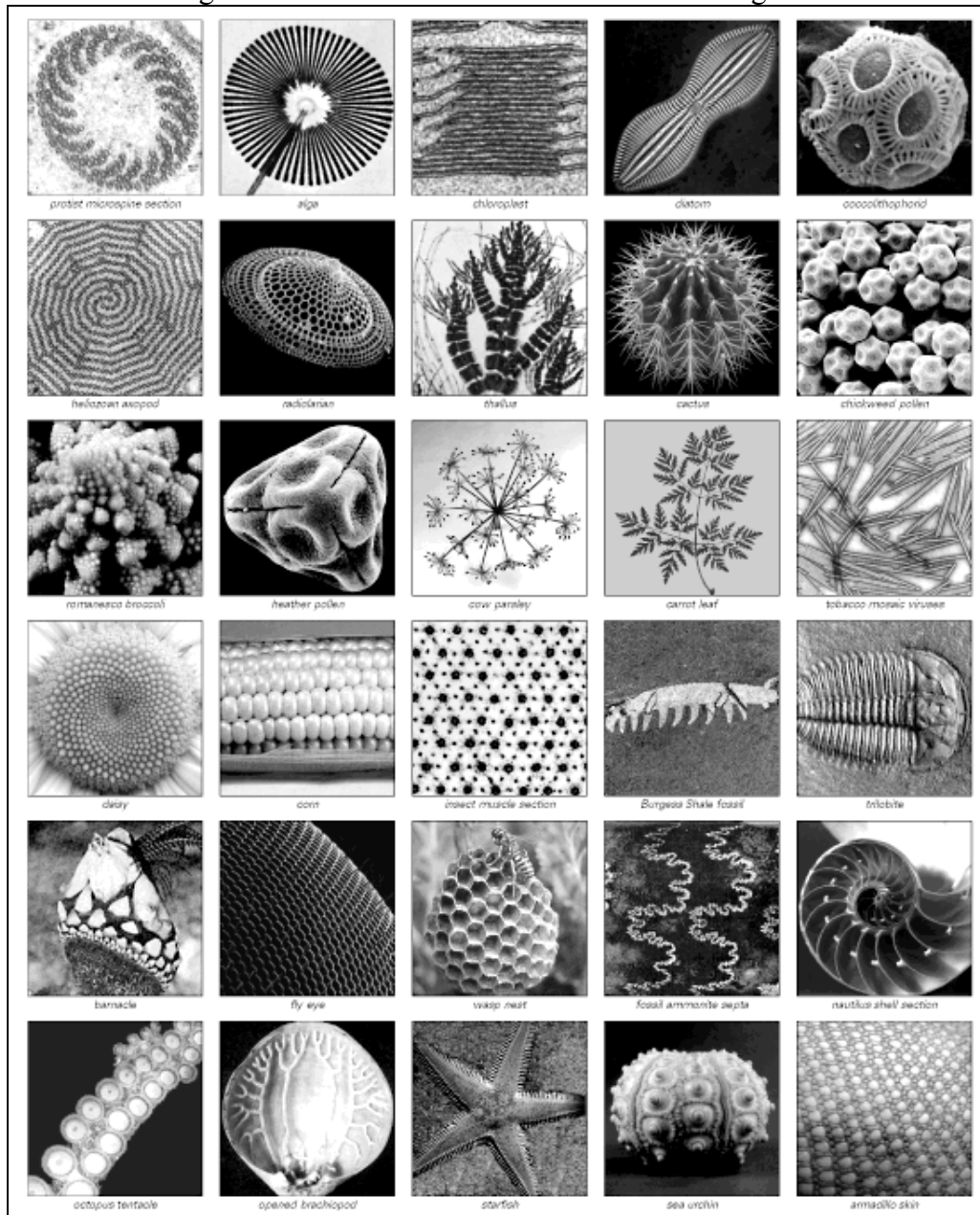
Figura 53 – Padrões de pigmentação encontrados em animais



Fonte: Wolfram (2002, p. 426).

Não limitado à pigmentação, a formação e o crescimento de plantas e animais também revelam padrões e características regulares suficientes para assumir que tais aspectos podem ser obtidos por programas simples (WOLFRAM, 2002). A Figura 54 apresenta estruturas animais e vegetais nas quais padrões podem ser observados.

Figura 54 – Padrões em estruturas animais e vegetais

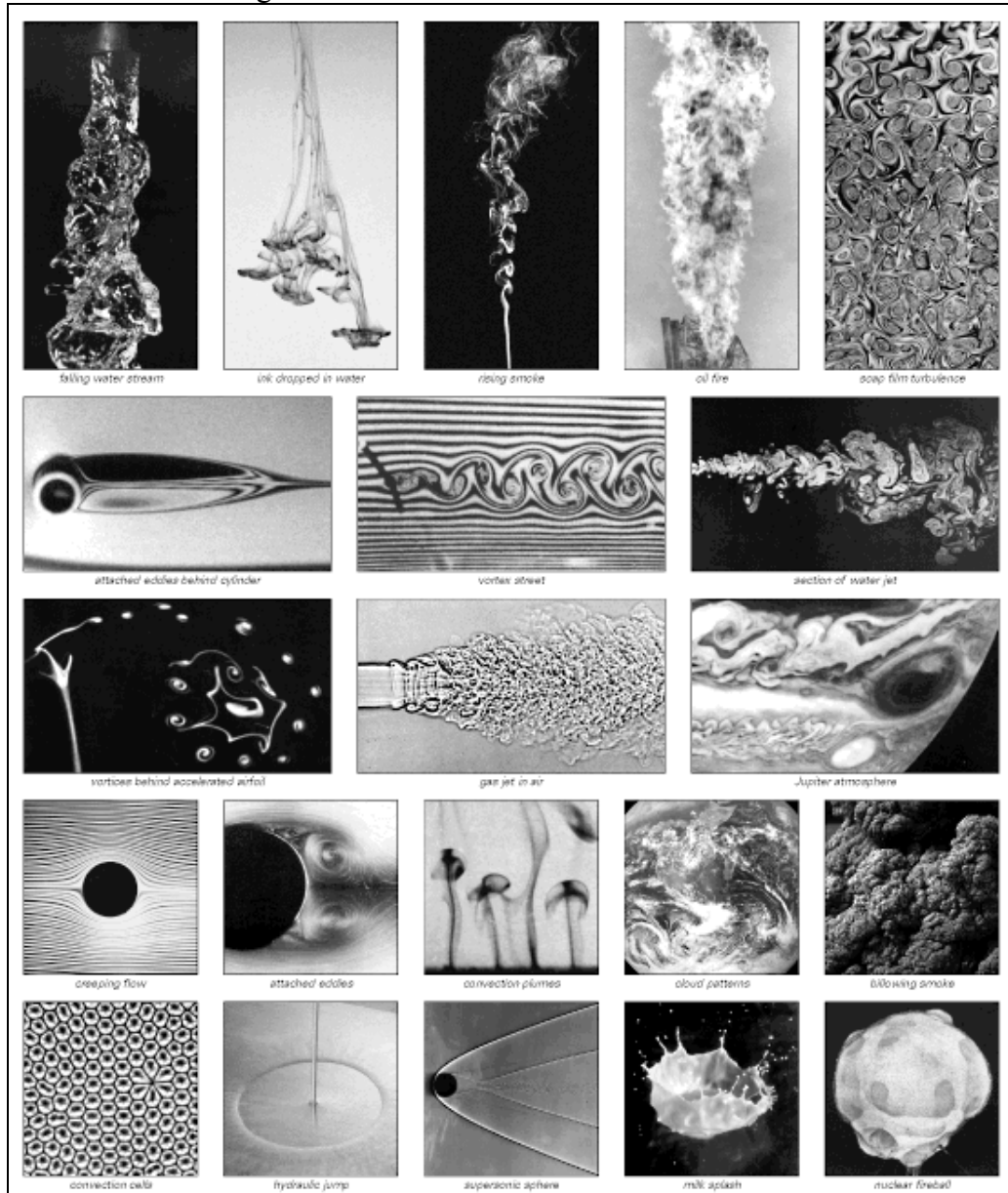


Fonte: Wolfram (2002, p. 385).

APÊNDICE C – Padrões e turbulência em fluidos

O comportamento de fluidos pode ser simulado através de autômatos celulares bidimensionais, conforme mostrado nas Figura 20 e Figura 21. A Figura 55 apresenta padrões típicos gerados por vários tipos de fluxos de fluido. É possível notar a ocorrência frequente de turbulência aparentemente aleatória.

Figura 55 – Padrões e turbulência em fluidos



Fonte: Wolfram (2002, p. 377).