Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

O uso de algoritmos emergentes na criação de arte através de criatividade computacional

Vitor Sousa Silva

Proposta de Trabalho de Graduação
Orientador: Prof. Dr. Filipe C. de A. Calegario
Área(s): Criatividade computacional

Recife
23 de maio de 2023

Universidade Federal de Pernambuco
Centro de Informática

Vitor Sousa Silva

O uso de algoritmos emergentes na criação de arte através de criatividade computacional

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Prof. Dr. Filipe C. de A. Calegario*

Recife
23 de maio de 2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Silva, Vitor Sousa.
O uso de algoritmos emergentes na criação de arte através de criatividade computacional / Vitor Sousa Silva. - Recife, 2023.
41p : il., tab.

Orientador(a): Filipe Carlos de Albuquerque Calegario
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2023.

1. Algoritmos. 2. Autômatos. 3. Criatividade. 4. Geração procedural. 5. Jogos. I. Calegario, Filipe Carlos de Albuquerque. (Orientação). II. Título.

000 CDD (22.ed.)

O caos é uma ordem por decifrar
—JOSÉ SARAMAGO (Homem Duplicado)

Agradecimentos

Gostaria de agradecer primeiramente a minha família, que me deu todo o suporte necessário para chegar até aqui. Agradeço a minha mãe, Sonali, que lutou muito para garantir que nada me faltasse; a meu avô, Waldemir, por ter sido meu exemplo moral, minha avó, Dulce, por me dar sempre os melhores conselhos. Aos meus tios (Waldemir e Rodrigo) e tias (Edna, Carol, Juliana e Raquel) que ofereceram pilares para que eu pudesse me inspirar desde muito antes de ingressar no curso. Aos caroneiros Breno e Fábio, pelas idas e vindas de Caruaru ao Recife.

Ao meu professor e orientador, Filipe Calegario, de quem tive a oportunidade de ser monitor-chefe por dois anos na disciplina de Sistemas Inteligentes, atividade esta que me deu experiência, me ingressou ao tema e fundamentou este trabalho.

A minha turma, que durante esses seis anos de curso foram minha família em Recife. Foram colegas como Lucas e Marvin, meus sócios, que me ajudaram em tantos projetos e trabalhamos em tantas empreitadas trilhadas juntos. Parceiros como Josué, Kristian e Mendonça (que me confiou a ser padrinho de casamento) e outros tantos que juntos tiramos do papel a ideia campeã de projetão do Flora e nos rendeu um ingresso na E.S.T.U.F.A e premiação na SBESC. A manos e manas como Teixeira, Bruno, Luana, Melissa e Íris por transmitirem alegria nos corredores do centro (e jogando os piores jogos de navegador possível). Aos colegas de podcast: Rodrigo, Ruy, Christian, Arthur, Adriano, Felipe e Berg por tornarem orgânico o ambiente e vivência universitária. Aos amigos mais centrados, Leão e Pedro, que transmitiram a mim visões mais equilibradas e sábias durante essa jornada. A pessoas como Luna, que me convenceram do propósito do código open-source. Parceiros como Zé e Felipe, que me encorajaram a alcançar a vaga de estágio no Google, quando nunca pensei que seria capaz. A JP, Hiro, Mari, Thamy, Thalles e Tamae, meus melhores amigos e amigas que descobri lá, por me mostrarem que o ambiente de trabalho perfeito se faz com as melhores pessoas.

Tudo isso não seria possível sem a existência da Universidade Federal de Pernambuco e do Centro de Informática. Foi através desta instituição que consegui meu primeiro estágio (e emprego), na Superintendência de Tecnologia da Informação, ao lado de Fernanda e Linaldo, colegas incomparáveis.

A estrutura de ponta do centro e qualidade incomparável dos professores (Castor, Sílvio, Sérgio, Santa Cruz, Ricardo, Eudes, Paguso, Gustavo, Ruy, Manoel, Carelli, Carlos, Adriano, Pasg, Germano, Francisco, Ricardo, Márcio, Giordano, Borba, Mário, Cristiano, Luciano, Geber, Alex Sandro, Paulo, Henrique, Augusto, Robson, Fábio e Jaelson) e professoras (Anjolina, Kátia, Carina, Renata, Patrícia, Teresa, Maíra) que me capacitaram durante essa jornada a ser o profissional que sou hoje.

Por fim, agradeço a Deus por ter me dado tudo que agradeci e tudo que jamais poderei agradecer.

Resumo

Algoritmos emergentes simulam interações entre agentes simples e o ambiente em que estão contidos. Os resultados das interações entre agentes são naturais e muitas vezes imprevisíveis. Podemos citar, por exemplo: agrupamentos espontâneos, criação de estruturas particulares e interações com o ambiente de forma especial. Algoritmos como esses são úteis na simulação de ecossistemas e na visualização de cenários complexos, tendo proximidade com tópicos relativos à inteligência artificial e algoritmos bioinspirados. Além disso, técnicas empregadas na construção de algoritmos emergentes podem ser utilizadas na criatividade computacional na produção de arte em seus diversos formatos (músicas, imagens, jogos, animações, etc), de simulações e de ambientes espontâneos. Este trabalho tem como objetivo realizar uma revisão integrada sobre algoritmos emergentes e sua relação com a criatividade computacional, bem como ilustrar uma através de implementação como podem ser utilizados autômatos celulares na geração de conteúdo artístico.

Palavras-chave: Algoritmos emergentes, autômatos celulares, criatividade computacional, geração procedural de conteúdo.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

CA	Cellular Automata: autômato celular.
GPU	Graphics Processing Unit: placa de processamento gráfico utilizada em computadores.
PCG	Procedural Content Generation: geração procedural de conteúdo.
PRNG	Pseudo Random Number Generator: gerador de número pseudo randômico.
RGB	Red Green Blue: Vermelho Verde Azul.
RPG	Role-Playing Game: jogo de interpretação de papeis.
RTS	Real Time Strategy: gênero de jogo de estratégia em tempo real.

Introdução

Autômatos celulares são estruturas celulares organizadas em grades e que evoluem em etapas discretas de acordo com um conjunto de regras baseado no estado de células vizinhas [3]. As células por si só são implementadas de forma bastante simples e possuem apenas alguns tipos e capacidades (como interação células vizinhas, funções de deslocamento ou ainda mudança de seu estado atual). As interações entre autômatos e ambientes podem ser utilizadas para visualizar o comportamento de algoritmos emergentes [4]. O termo "emergente" é adotado quando as entidades observadas demonstram características particulares apenas após interagirem com outras partes, isto é, os agentes não apresentam o mesmo comportamento quando isolados. Neste trabalho são discutidas abordagens emergentes para criação de criatividade computacional. É realizada uma revisão literária com propósito de situar o leitor sobre trabalhos e ferramentas distintas (autômatos, sistemas de Lindenmayer, fractais e algoritmos genéticos) usadas para alcançar objetivos particulares. Como forma de demonstração, será produzida uma ferramenta autômata para auxiliar desenvolvedores de jogos na geração procedural de mapas para *videogames*. Os objetivos foram traçados a partir dos trabalhos futuros identificados nos trabalhos futuros do texto de Ziegler [5]. Os objetivos dessa técnica são comparados com outra abordagem emergente semelhante que utiliza algoritmos genéticos [2], as métricas comuns são fundamentadas através de estudos geomorfológicos de outro trabalho [6]. Por fim, são comentadas as conclusões a respeito dos resultados (viabilidade, vantagens e desvantagens) e feitas sugestões para explorar e complementar esse trabalho.

Revisão literária

2.1 Fundamentos teóricos

2.1.1 Informática Teórica

A informática teórica é um ramo da computação que estuda diversos campos relacionados a aplicações matemáticas e lógicas na ciência da computação, alguns objetos de seu estudo são: algoritmos (estudos de computabilidade e complexidade), estruturas de dados (modelagem de sistemas complexos), criptografia (e temas voltados a segurança), teoria dos números, teoria dos jogos; dentre outros tópicos. Um de seus objetos de estudo é a teoria dos autômatos.

2.1.2 Autômatos

O significado do termo *autômato* é de origem grega, significa "agindo por vontade própria", e já foi historicamente atribuído a diversas invenções capazes de realizar ações automáticas. Por vezes, estas criações são semelhantes a seres humanos ou outros organismos. No desenvolvimento de jogos, autômatos são utilizados para simular comportamentos que enriquecem a imersão em ambientes e agentes. Na computação, autômatos são fundamentais para entender o comportamento lógico de uma máquina computacional: uma estrutura com capacidade de interpretar um alfabeto pré-definido, estando organizada em um número finito de estados, e que possui funções que permitem alternar o estado atual, podendo ou não, ao fim da entrada, apresentar um resultado.

2.1.2.1 Autômatos celulares

Autômatos celulares são um tipo de autômato organizado em uma grade (sem tamanho pré-definido) dividida em unidades chamadas de células. Assim como um autômato tradicional, cada célula possui um número finito de estados e funções que interagem com o estado atual, porém, as funções exercem um papel fundamental na interação com as células vizinhas. A execução de um CA não ocorre pela leitura de uma entrada (como um automato tradicional), mas sim por sucessivas aplicações de uma função global fixa a um dado estado inicial, a partir disso, o automato "ganha vida".

2.1.3 Sistemas de Lindenmayer

Sistemas de Lindenmayer, também conhecidos por sua abreviação em inglês *L-systems*, são sistemas gerados a partir de uma teoria axiomática elaborada pelo biólogo Aristid Lindenmayer

em 1968 no artigo *Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs*, publicado no 18o volume do jornal de biologia teórica. Os sistemas são baseados na aplicação de regras de escrita que produzem sequências de estruturas multicelulares. As estruturas descritas por Lindenmayer começam a ser representadas através de caracteres atômicos e, a partir da aplicação das regras, vão moldando novas cadeias, as quais serão suscetível a outras transformações. Lindenmayer desenvolve sua teoria partindo de modelos gramaticais para lineares, à grafos, seguidos de um modelo bidimensional, e por fim demonstrando representações tridimensionais.

2.1.4 Fractais

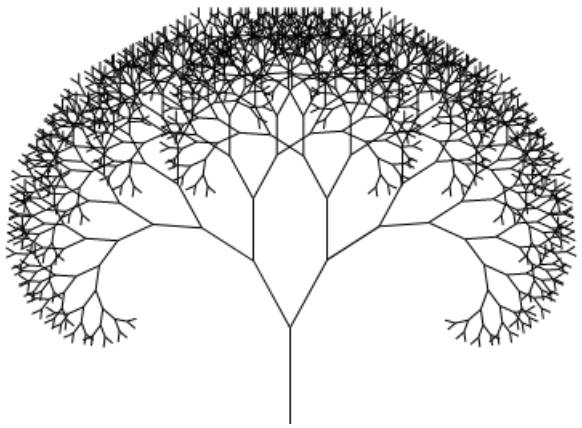

Figura 2.1 Fractal semelhante a uma árvore, Wikipédia.

Fractais são estruturas geométricas estudadas na matemática e que podem ser encontradas em outras áreas, como: arte, biologia, arquitetura e tecnologia; tendo papel fundamental na teoria do caos. A definição concreta do conceito matemático que identifica um fractal é complexa, porém, suas características são bastante claras e únicas. A "auto-semelhança"é uma dessas particularidades: não são observadas mudanças em sua estrutura, não importa o quão for aumentado o nível de detalhamento (*zoom*) em um fractal; isto resulta na característica de "nenhuma-parte-diferenciável", pois normalmente, pode-se verificar o tamanho de uma estrutura analisando a menor de suas partes, entretanto, por suas dimensões serem recursivamente infinitas, esse traço não se aplica a um fractal. Na imagem acima, podemos verificar a repetição de um padrão semelhante a um "Y"(haste que se divide em duas) e resulta numa estrutura visualmente próxima a uma árvore.

2.2 Implementações

Algumas implementações que representam bem a utilização de autômatos e estruturas semelhantes para produção de conteúdo criativo.

2.2.1 Conway's Game of Life

Figura 2.2 Captura de estado inicial aleatório do *Conway Game of Life*, autor.

Game of life, em português: "jogo da vida", foi um jogo desenvolvido por John Horton Conway em 1970, utilizando a linguagem de programação Pascal, sendo este o exemplo mais conhecido de um automato celular [7]. O jogo consiste em uma grade dividida em quadrados (chamadas de células) de mesma dimensão os quais possuem dois estados: desativados (identificados pela cor branca), ou ativados (identificados pela cor preta). Cada célula pode interagir apenas com as outras 8 células de sua vizinhança (isto é: vizinhos horizontais e verticais bem como as diagonais). O jogo não necessita de um jogador para interagir com ele, sendo sua evolução determinada somente pela sua configuração de estado inicial. Entretanto, existe a possibilidade do jogador interagir com o jogo: ele pode modificar a configuração inicial ou em interferir em qualquer iteração seguinte, o jogo então aplicará as regras, de forma idêntica a sua execução automática. As regras gerais do Game of life podem ser sintetizadas como [8]:

- Subpopulação: qualquer célula viva com menos de dois vizinhos se tornará uma célula morta.
- Qualquer célula viva com dois ou três vizinhos permanece viva (equilíbrio).
- Superpopulação: qualquer célula com mais de três vizinhos vivos se tornará uma célula morta.
- Reprodução: qualquer célula morta com exatos três vizinhos se tornará uma célula viva.

2.2.2 Lenia

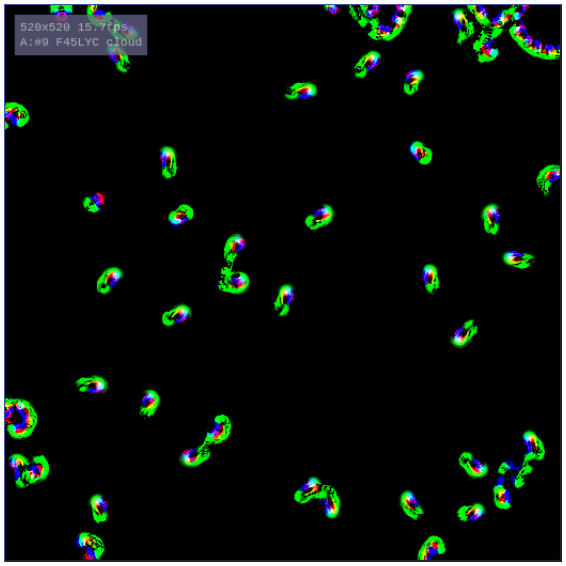

Figura 2.3 Captura do *Lenia* após algumas iterações, autor.

Lenia é um projeto inspirado no *Conway's Game of Life*, ele foi desenvolvido em 2015 originalmente com o nome de *Primordia*, utilizando da linguagem de programação *JavaScript* [9]. A principal diferenciação entre Lenia e sua inspiração original é o uso de espaços contínuos. No novo ambiente em questão os autômatos não habitam mais a grade celular definida discretamente. Isso expande as possibilidades de interação e organização entre os autômatos, bem como permite a visualização de simulações mais ricas em detalhes gráficos. O projeto já foi portado, re-escrito e expandido para outras linguagens de programação como *MathLab, Python* (onde versões 3D e 4D foram desenvolvidas e outras particularidades descobertas), *WebGL* (versão otimizada para lidar com processamento feito por placas de processamento gráfico dedicado (GPUs) e que está ilustrada na imagem acima).

2.2.3 Jogos de areia caindo

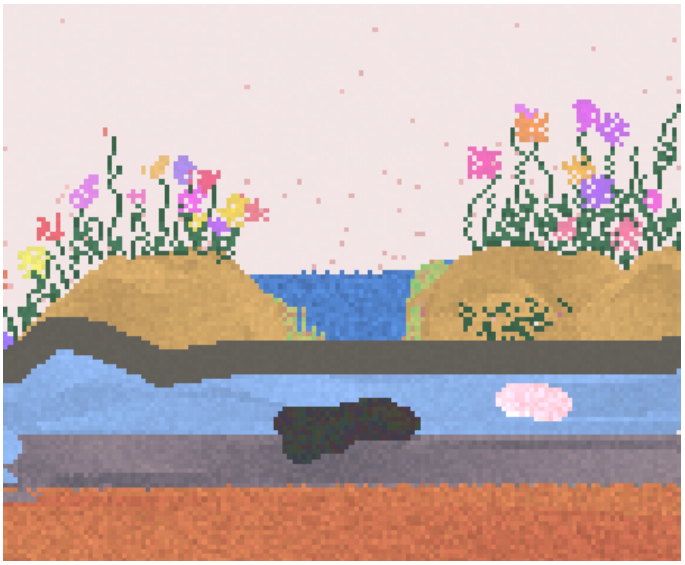

Figura 2.4 Captura do *Sandspiel* simulando ecossistema, autor.

Jogos de areia caindo, mais facilmente identificados pelo termo em inglês, *falling-sand games*, são implementações que simulam ecossistemas utilizando autômatos. A representação da areia é feita em pixels que podem ser controlados pelo usuário. De forma semelhante às células utilizadas no *Conway's Game of Life*, ao adicionar ou remover grãos ocorrem interações com seus vizinhos, sendo o grande diferencial a quantidade distinta de interações e tipos de grãos (identificados cada classe por sua cor). Essa forma criativa de demonstração de um ambiente é sem dúvidas mais atrativa que as anteriores por ser visualmente agradável e dar ao usuário o controle do ambiente de forma intuitiva. Na imagem acima está uma captura do jogo *Sandspiel*, desenvolvido no fim de 2018 pelo artista computacional Max Bitkker. O jogo foi programado utilizando JavaScript, Rust e WebAssembly e está disponível gratuitamente para navegadores. Um de seus diferenciais é possuir 20 elementos que interagem entre si e permitem ao jogador simular um ecossistema com bastante riqueza criativa. Como descrito em publicação que explica sua trajetória artística e detalhes de desenvolvimento do *Sandspiel*, este é o terceiro (ou quarto) projeto de Bittker envolvendo a temática *falling-sand*, o primeiro foi desenvolvido em 2015 [10]. Algumas interações de autômatos e seus resultados que ocorrem em *Sandspiel* são: queima quando óleo, gás, madeira, plantas, sementes ou fungos entram em contato com lava, fogo; disseminação de gelo, plantas, fungos ou sementes quando entram em contato com água; formação de pedra a partir do contato de lava e água, dentre outras interações. O código do projeto é aberto e continua sendo mantido por Max no seu GitHub.

2.3 Aplicações em contextos distintos

2.3.1 Uso na arquitetura e urbanismo

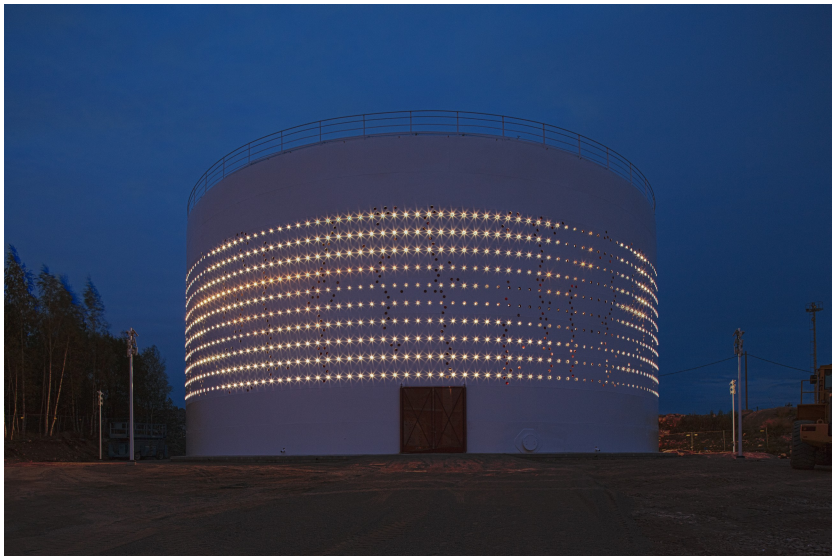

Figura 2.5 Silo 468 feito por *Lighting Design Collective*, Tapio Rosenius.

A aplicação de algoritmos emergentes utilizando sensores para promoção da criatividade na iluminação pública criativa foi um dos tema explorado no artigo *Design possibilities of emergent algorithm for adaptive lighting system*, desenvolvido em conjunto por universidades e arquitetos de países nórdicos. O estudo envolveu analisar o uso de algoritmos emergentes, sistemas de Lindenmayer e algoritmos de enxame com o objetivo de otimizar recursos da iluminação pública. A proposta de iluminação urbana adaptativa do artigo, através da observação dos algoritmos citados, visa criar formas criativas de proporcionar experiências artísticas e estéticas além de poupar recursos elétricos, mas sem deixar pessoas no escuro [11]. Para realizar este trabalho foi proposto utilizar da ubiquidade e da computação penetrante para desenvolver esta ambiente inteligente. Dentre os requisitos funcionais, foi necessário elencar a necessidade de reservar uma interface para que designers e arquitetos interfiram no comportamento dos algoritmos de forma a atingir comportamentos artísticos esperados. A imagem acima é uma captura do Silo 468: uma peça arquitetônica localizada em Helsinque, Finlândia, que utiliza da tecnologia de enxames para controlar sua iluminação. O velho silo foi reformado para servir de estrutura decorativa, ele possui 2012 aberturas (450 com espelhos de metal com posição controlada pelo vento) e sua pintura de cor discreta possibilita uma maior apreciação visual das luzes controladas por um algoritmo. O controle é feito utilizando o *framework* OpenFrameworks feito em C++, arcabouço (framework) o qual é otimizado para desenvolvimento de criatividade computacional [12].

2.3.2 Uso em produção sonora

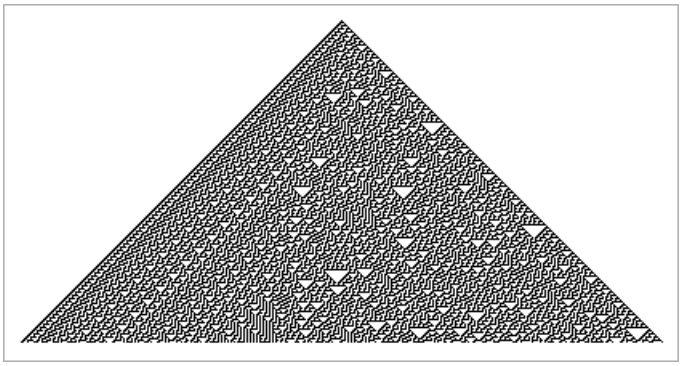

Figura 2.6 Captura das 100 primeiras iterações do autômato gerado pela regra 30, WolframTones.

No artigo: *Interactive sonification exploring emergent behavior applying models for biological information and listening*, é proposta a criação de um *framework* que possibilite a transformação de dados em som. O processo, descrito como sonificação, é um esforço para produzir e captar as nuances envolvidas no ambiente sonoro. Para testes são utilizados duas ferramentas capazes conhecidas por sua capacidade de emergir resultados inesperados algoritmo de enxame e um circuito elétrico do tipo de de Chua (*Chua circuit*). Um diferencial desta implementação frente às abordagens citadas anteriormente é a maior possibilidade de exploração dessa dimensão. A dimensão sonora tem características que permitem várias configurações únicas: altura, sonoridade, qualidade do tom, duração, silêncio, repetição e padrão [13].
WolframTones é uma aplicação desenvolvida pelo instituto *Wolfram Research* a partir das descobertas do homônimo, Stephen Wolfram, compiladas em seu livro *A New Kind of Science*, publicado em 2002. Em seus experimentos na década de 1980, Wolfram trabalhou com autômatos unidimensionais. Para explorar o potencial generativo desses autômatos, ele observou que, a partir de regras de geração de padrões, era possível gerar estruturas regulares e outras nem tanto. Das 256 chamadas regras elementais, a regra número 30 é a primeira a apresentar comportamento irregular [14]. Os fundamentos da produção musical do WolframTones utilizam da capacidade emergente dessas regras únicas: os estados do autômato atuam como identificadores de notas musicais e suas alturas. O procedimento de adaptação da representação do autômato em som consiste em: fatiar as laterais figura do autômato a partir do meio para obter largura 12 (número que representa quantidade de tonalidades musicais); rotacionar em 90 graus as figuras geradas pelo autômato; adicionar as estruturas fatiadas uma após a outra, a partir da primeira (desta forma é preservada a regularidade do autômato, e portanto, a musical). Após isto, a leitura feita pelo sintetizador deve tratar a figura como uma partitura, onde a células ativas indicam presença de um respectivo tom. O site permite que o usuário personalize o tipo de regra, a altura, instrumentos, escala, ritmo e duração. O WolframTones também mostra o autômato e regras que geraram a composição (como a da figura ilustrada acima), a qual pode ir até 30 segundos de duração, sendo possível salvar a melodia.

2.3.3 Uso em simulações físicas

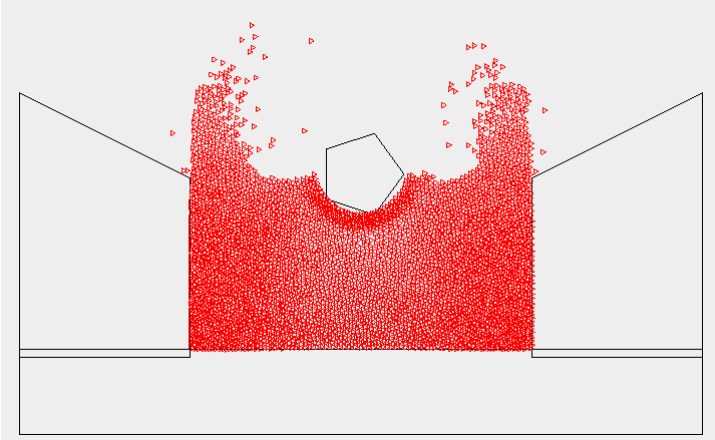

Figura 2.7 Captura do cenário *Particles* do *Liquidfun* [1], autor.

Capacidade emergentes podem ser utilizadas na experimentação de interação de partículas e seus comportamentos físicos resultantes. *Liquidfun* é um projeto implementado em várias linguagens de programação (C++, Java e JavaScript) de código aberto fornecido pelo Google que possibilita simulações de partículas em situações distintas [1]. Neste cenário, as partículas são os elementos que induzem comportamento emergente. Em algumas das simulações de demonstração é possível o usuário adicionar partículas e/ou interagir com objetos. As interações entre partículas permitem observar propriedades físicas como elasticidade, tensão, gravidade, empuxo, dentre outras. Na imagem acima, a captura foi realizada após o pentágono cinza ser solto (em queda livre) e entrar em contato com partículas vermelhas triangulares, ocasionando na transmissão da energia potencial gravitacional em energia cinética. Essa interação provoca pressão e contato elástico das partículas do sistema (agregado de triângulos), causando impulsão e transbordamento de alguns triângulos.
Os autômatos também podem ser utilizados para simular situações como modelos de evacuação de ambientes, que visam simular fluxo, comportamento e potenciais riscos envolvendo deslocamento de pessoas em ocasiões distintas. De forma semelhante, o comportamento no trânsito veicular também pode ser simulado, onde fatores ímpares como limites de velocidade e sinalização distinguem esse ambiente do citado anteriormente. A passividade do agente perante o ambiente é verificada no artigo que discute o impacto (positivo ou negativo) do vento e das marés na trajetória marítima de um navio.

2.3.4 Uso em criptografia

Por possuírem características que permitem dispersão e criação de aleatoriedade, autômatos podem ser utilizados na encriptação de imagens. A aplicação, descrita no artigo *RGB Image Encryption through Cellular Automata, S-Box and the Lorenz System* faz uso da regra número 30 (a mesma descoberta por Wolfram, citada e usada na subseção: "Uso em produção sonora"), que segundo o artigo, pode ser considerada uma geradora de números pseudoaleatório (em

seu acrônimo em inglês, *PRNG*). A observação que se faz é relacionada a criptografia: para calcular-se o estado do autômato da regra 30 a partir do estado inicial é uma tarefa simples; entretanto, encontrar o passo anterior a partir de um dado estado qualquer é uma atividade complexa. Essa complexidade computacional inerente permite utilizar o autômato da regra 30 como cifra de uma das chaves públicas usada para codificar a imagem [15].

2.3.5 Uso em predição de cenários

Autômatos celulares contribuem em pesquisas relacionadas a predição e modelagem de cenários futuros. Alimentados com dados de séries temporais e auxiliados por abordagens específicas relativas ao campo de estudo, autômatos servem de ferramenta para geração de possíveis desfechos. No Brasil, trabalhos como *Previsão da demanda hídrica de Fortaleza por meio de autômatos celulares* ou ainda *Simulação de cenários urbanos por autômato celular para modelagem do crescimento de Campinas – SP , Brasil*, utilizam de séries temporais para compreensão da evolução do ambiente estudado. O processo de início lida com a transformação de cada mapa em uma matriz, então, cada célula da matriz é atribuída a uma classe estudada; logo, a partir da evolução celular observada e da verificação estatísticas relacionadas ao objeto de estudo (para calibração do modelo), o autômato é capaz de gerar novos estados (predições) do que foi modelado.

2.4 Aplicações em desenvolvimento de jogos

A geração procedural é uma técnica antiga e que foi bastante utilizada em períodos onde as capacidades de *hardware* (principalmente memória/armazenamento) eram limitadas. Seu objetivo é possibilitar uma maior variabilidade de: ambiente, agentes, equipamento, ou outros componentes de um jogo. No passado, a solução encontrada foi explorar da capacidade de processamento (que também era escassa) para adicionar conteúdo gerado em tempo de execução. As abordagens mais comuns incluem o uso de *seeds* (sementes, em tradução livre), valores normalmente numéricos que, aplicadas em algoritmos matemáticos, implementam diversidade através de aleatoriedade ao conteúdo jogado. Atualmente, os componentes eletrônicos de computadores, *smartphones* e consoles (dispositivos para jogatina) não carecem mais do uso de geração procedural para otimização de recursos, dado os robustos *hardwares* disponíveis a baixo custo no mercado. Entretanto, a geração procedural tem um papel importante na indústria de jogos, pois permite que desenvolvedores de menor porte (popularmente conhecidos como indies) consigam expandir a duração da jogatina e quantidade cenários. Essa técnica os coloca, de certa forma, em pé de igualdade em comparação com desenvolvedoras de médio-grande porte, pois reduz o tempo de desenvolvimento e, portanto, os impactos no orçamento. Alguns exemplos de jogos e seu uso da geração procedural são:

- Minecraft Dungeons e No Man's Sky (geração de mapa)
- Binding of Isaac (posicionamento de itens e inimigos no mapa)
- Borderlands 2 (randomização de equipamento)

Objetivos

Neste trabalho serão apresentados conceitos de autômatos celulares e projetos existentes na área de criatividade computacional que foram realizados utilizando os mesmos. Através da elaboração de uma biblioteca de geração de mapas para jogos com auxílio de autômatos celulares, será demonstrado como é o processo de implementação de um automato celular visando aplicação na área de criatividade computacional (geração procedural de mapas para jogos do estilo RTS). Esta biblioteca será implementada utilizando fundamentos e sugestões de trabalhos futuros descritos num trabalho criado e validado através de partidas disputadas entre jogadores em mapas gerados usando autômatos [5]. Para quantificar os resultados atingidos e poder compará-los com outras implementações, são utilizadas métricas abordadas por um artigo que estuda o mapas gerados por algoritmos genéticos [2]. O objetivo de ambas as obras utilizadas como fundamento (e também deste trabalho) é o mesmo: gerar mapas de qualidade que despertem interesse e retenham jogadores ao jogo. Este exemplo será construído após pesquisa e estudo da literatura existente, sendo abordados também alguns tópicos tangentes, como: informática teórica e algoritmos genéticos. Também é de interesse desse trabalho enriquecer a literatura dessas áreas ao compilar técnicas utilizadas na produção criatividade computacional.

Metodologia

A implementação presente neste artigo é resultado da pesquisa e revisão da literatura existente. A ferramenta e métodos empregados tentam atingir, através das métricas de qualidade apresentadas, um gerador de mapas quantitativo para ser reutilizado por desenvolvedores de jogos. Em princípio o domínio específico do mapa é para jogos RTS, pois foi fundamentado utilizando literatura descrita para tal. As amostras de mapas foram geradas pela ferramenta, assim como o cálculo das métricas estarão implementadas junto à ferramenta. A análise da qualidade de um mapa gerado é feita com base em métodos descritos em artigos pré-existentes (a descrição aprofundada do método está na seção de desenvolvimento). Algumas das limitações existentes são: falta de coleta de opinião de desenvolvedores de jogos e a falta de revisão de especialistas sobre os padrões implementados em *Unity* e *C#*. Não existem conflitos de interesse relacionados a este trabalho. O cerne do método visa produzir uma ferramenta que siga os princípios validados com o público de Ziegler [5] e competitiva em relação a implementação baseada em algoritmos genéticos de Frade [2]. Então, a confiabilidade da ferramenta desenvolvida é baseada em alguns de seus requisitos (funcionais e não funcionais):

- Possuir poucos requisitos de *hardware* para ser utilizada: sendo idealmente executada num *desktop* tradicional;
- Possuir funcionalidades simples de serem executadas: levar poucos passos para usuário conseguir gerar mapa desejado;
- Permitir exportação seus resultados (mapas) em formatos (de imagem ou de dados) livres e universais: para poderem ser utilizados por outros *softwares*, como *Blender, Unity, Godot*;
- Permitir personalização de parâmetros;
- Capacidade de apresentar/exportar resultados: para garantir confiabilidade ao usuário;
- Ser open-source e permissiva a modificações;

Desenvolvimento

Neste capítulo é descrito o processo de desenvolvimento da ferramenta. Partindo da introdução (onde são descritos os objetivos e as tomadas de decisão que impactaram no código a ser desenvolvido), para análise do código (onde o código é exposto e comentado em seus pormenores) que serve de fundamento para compreensão aprofundada das variáveis do capítulo de resultados. Durante o desenvolvimento, para exemplificar de forma direta e não restritiva quanto a implementação concreta, as sugestões de pseudocódigo foram feitas numa linguagem semelhante a Python.

5.1 Introdução e trabalhos relacionados

Após realizar a leitura e revisão da literatura existente, foram analisados projetos concluídos, a fim de ser compreendido o contexto em que se encontra o estado da arte. A ideia e motivação inicial era elaborar uma ferramenta para geração de mapas no estilo de *dungeons* (arcabouços ou masmorras) utilizando CAs. Esse tipo de mapa é bastante popular em jogos de *RPG*, sua estrutura normalmente simula aparência de minas: tem organização labiríntica, podendo conter áreas largas interligadas por conexões estreitas (que representam túneis). Entretanto, já existem abordagens na literatura que exploraram muitas possibilidades utilizando esse modelo de geração e esta mesma aplicação [16].

O foco da minha proposta de exploração e desenvolvimento foi definido após leitura dos trabalhos futuros do artigo *Generating Real-Time Strategy Heightmaps using Cellular Automata* (em tradução contextualizada: "geração de mapas de altura para jogos de estratégia em tempo real utilizando autômatos celulares"), pois há uma diferenciação na implementação do autômato tradicionalmente utilizado (os autores geram vários mapas a partir de autômatos e cada mapa é sobreposto para geração de um mapa final). As sugestões dos autores para trabalhos futuros são baseadas em modificações do que foi realizado no artigo. Tendo essas propostas em mente, foi necessário adaptar os critérios avaliativos da implementação deles: o artigo em questão utiliza como métrica de seus experimentos a satisfação de jogadores do jogo *Supreme Commander* para com o mapa gerado. Não houve como avaliar empiricamente os mapas gerados pela minha implementação com jogadores, então, para não tornar um fator limitante para outras pesquisas, decidi sair do escopo do jogo utilizado por este artigo. Ainda assim, minha proposta de avaliação parte da conclusão de uma das respostas coletadas no artigo: mapas com tipo de terreno diverso e balanceado (proporcionalidade entre áreas de biomas e acessibilidade de terrenos) são fatores bem avaliados pelos jogadores. Entretanto, nenhuma métrica para avaliação deste fator é descrita no artigo, então, para avaliar a qualidade de um mapa foi necessário

utilizar de alternativas descritas em outro artigo para serem utilizadas como métrica.

No artigo *Automatic evolution of programs for procedural generation of terrains for video games: Accessibility and edge length constraints* (em tradução livre: evolução automática de programas para geração procedural de terrenos para vídeo-games: restrições de acessibilidade e comprimentos de bordas), são descritas técnicas utilizadas para quantificar e avaliar a qualidade de mapas gerados através da técnica de algoritmos genéticos [2]. A qualidade de um mapa, neste artigo, tem relação com a percepção de apelo visual que um jogador teria para com o mapa desenvolvido. As variáveis utilizadas para calcular o *fitness* (função matemática que define o quão bem é um indivíduo produto de algoritmo genético) são acessibilidade e o comprimento de borda. As variáveis estão relacionadas da seguinte forma: um mapa completamente acessível é desinteressante ao jogador, dado que é um mapa plano; o apelo visual de um mapa contendo muitos acidentes geográficos terá grande um grande valor para variável comprimento de bordas, mas prejudicará a exploração e viabilidade de deslocamento do jogador.

Então, o desenvolvimento deste trabalho foi fundamentado ao associar a técnica e sugestões de implementação do primeiro artigo citado as abordagens de avaliação do segundo artigo.

Em seguida, foi escolhido o motor de jogo (em inglês, *game engine*) Unity para ser desenvolvida a técnica de PCG. Os principais fatores levados em consideração na escolha da *engine* foram: a grande comunidade de desenvolvedores e de projetos construídos em Unity, possibilidade de exportação multiplataforma do Unity (estão disponíveis: dispositivos móveis, computadores, consoles, aparelhos de realidade aumentada/virtual, etc.) e também a complexidade de implementação; pois a intenção é que seja útil para a maior quantidade de desenvolvedores possível e o código possa ser adaptado, extensível e compreensível aos leitores.

5.2 Sumário de objetivos

Como forma enumerar e facilitar a análise de conteúdo desenvolvido, os itens abaixo foram traduzidos da seção de trabalhos futuros do artigo *Generating Real-Time Strategy Heightmaps using Cellular Automata Generating Real-Time Strategy Heightmaps using Cellular Automata*:

1. Experimentar adicionar outras regras de vizinhança, como a de Moore ou outros tipos diferentes de vizinhança.

2. Experimentar adicionar mais estados ao autômato (além dos tradicionais ativado e desativado, ou vivo/morto) de forma a gerar todo o mapa em apenas uma camada.

Entretanto, a abordagem apresentada no artigo desconsidera diretamente o apelo visual do mapa, os autores avaliam o critério de "*fun*"(diversão) para jogadores vencedores e perdedores de uma partida utilizando o mapa gerado. Esse fator é apresentado em [2] como "apelo visual", e segundo ele, influencia positivamente na avaliação do jogador. Minha implementação tenta utilizar da função *fitness* apresentadas no em [2] como quantificador do apelo visual, mas saindo do escopo de algoritmos genéticos e trazendo implementação para a área de autômatos celulares. Através dessa adaptação, por não necessitar de uma da mesma quantidade de detalhes de implementação utilizada no algoritmo genético, utilizando da mecânica de regras simples de autômatos, tenta-se atingir bons resultados.

5.3 Métricas

5.3.1 Acessibilidade

No artigo *Automatic evolution of programs for procedural generation of terrains for video games: Accessibility and edge length constraints* as métricas utilizadas para otimizar a função de *fitness* do algoritmo genético são baseadas em cálculos topográficos Horn(1981) para avaliar o declive de um terreno. De forma a simplificar a compreensão, a altura é calculada como eixo Z (que denomina as variáveis da tabela abaixo).

z1	z2	z3
z4	z5	z6
z7	z8	z9

Considerando a organização celular do autômato de tamanho 3x3 da tabela acima, os autores do artigo *Hill shading and the reflectance map, Horn(1981)* utilizam de técnicas para calcular o declive (*Slope*: fórmula 5.1) percentual em relação a célula z5.

$$Slope(\%) = 100 \times \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{5.1}$$

As derivadas em relação a x e y são calculadas visando encontrar a inclinação deste bloco de 9 células. Para tal, é realizada a subtração da altura (eixo Z) em ambos os eixos (leste-oeste: fórmula 5.2; norte-sul: fórmula 5.3), desconsiderando a célula central z5, isso justifica as variáveis z2, z4, z6 e z8 (todos vizinhos imediatos de z5) sempre encontrarem-se duplicados em seus cálculos (para compensar a ausência de z5).

$$\frac{\partial f}{\partial x} \approx \frac{z_3 + 2z_6 + z_9 - (z_1 + 2z_4 + z_7)}{8\Delta x} \tag{5.2}$$

$$\frac{\partial f}{\partial y} \approx \frac{(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)}{8\Delta y} \tag{5.3}$$

A partir da aplicação das operações acima, teremos obtido um mapa A, de dimensões iguais aos do autômato celular, o qual identifica as células acessíveis como 0 e não-acessíveis como 1. Para calcular valor de acessibilidade do terreno, é necessário efetuar o produto das dimensões do terreno dividido pela maior quantidade de blocos acessíveis (maior área acessível contínua no terreno, A_+), tomemos v como esse percentual relativo a quantidade de células total.

$$v = \frac{largura \times altura}{A_+} \tag{5.4}$$

O método utilizado na minha implementação para encontrar o A_+ foi através do algoritmo de preenchimento de área (*flood fill*), que identifica o maior conjunto de elementos dada uma condição (no caso, se a célula é acessível), a partir de uma célula inicial. A_+ não pode ser zero.

Após utilizá-lo várias vezes, utilizando como origem todas as células do mapa, é atribuída a maior área como A_+. Algumas heurísticas podem ser utilizadas para reduzir o número de execuções: utilizando um conjunto é possível evitar que uma célula que já foi encontrada seja verificada novamente no *flood fill*; ou para a execução na célula atual, caso a área retornada ultrapasse 50%.

```
conjunto_visitado = {}
def Flood_Fill(x, y)
    se par(x,y) pertence a conjunto_visitado
        \    OU tamanho(conjunto_visitado) >
        \    tamanho(quantidade_celulas_automato)/2:
        retorne

    conjunto_visitado->adicionarPar(x,y)

    Flood_Fill(x-1, y-1)
    Flood_Fill(x, y+1)
    Flood_Fill(x+1, y)
    Flood_Fill(x+1, y+1)
    retorne
```

Para normalizar os valores, tomemos com v_S o valor de acessibilidade subtraído do limite desejado pelo usuário (v_t). O cálculo de v_t é definido pelo quociente entre o produto as dimensões de altura e largura e a função piso do mesmo produto multiplicado pelo limite de área acessível estabelecido pelo usuário, p_a (que varia de diferente de 0 até 1). A função piso serve para atingir a condição de parada do algoritmo genético.

$$v_S = |v - v_t| \tag{5.5}$$

$$v_t = \frac{largura \times altura}{\lceil largura \times altura \times p_a \rceil} \tag{5.6}$$

5.3.2 Comprimento de borda

De forma semelhante a elaborada acima, os cálculos realizados para obter-se valores de comprimento de borda são bastante semelhantes

- E: relação entre total de células e quantas estão numa bordas.
- E_+: indica a quantidade de células inacessíveis. E_+ não pode ser zero.
- E_t: limite de células inacessíveis desejado pelo usuário.
- p_e: percentual de comprimento de borda desejado em relação a área total (varia de diferente de 0 até 1).

- ∇^2f: representa o operador de Laplace aplicado ao mapa A (acessibilidade das células), definido na subseção anterior. Se o resultado da aplicação da fórmula for positivo, então o valor faz parte de uma borda.

$$E = \frac{largura \times altura}{E_+} \tag{5.7}$$

$$E_S = |E - E_t| \tag{5.8}$$

$$E_t = \frac{largura \times altura}{\lceil largura \times altura \times p_e \rceil} \tag{5.9}$$

$$\nabla^2 f \approx 8z_5 - (z_1 + z_2 + z_3 + z_4 + z_5 + z_6 + z_7 + z_8 + z_9) \tag{5.10}$$

Por fim, o *fitness* é regulado através do uso de 2 pesos w_a e w_e:

$$fitness = w_a * v_s + w_e * E_s \tag{5.11}$$

5.4 Implementação dos objetivos

5.4.1 Regras de vizinhança

Em [2], os experimentos são realizados utilizando a vizinhança de Von Neumann para avaliar o estado dos autômatos vizinhos. Na imagem abaixo, estão ilustradas as diferenças entre as vizinhanças implementadas. O referencial é o quadrado central (x,y).

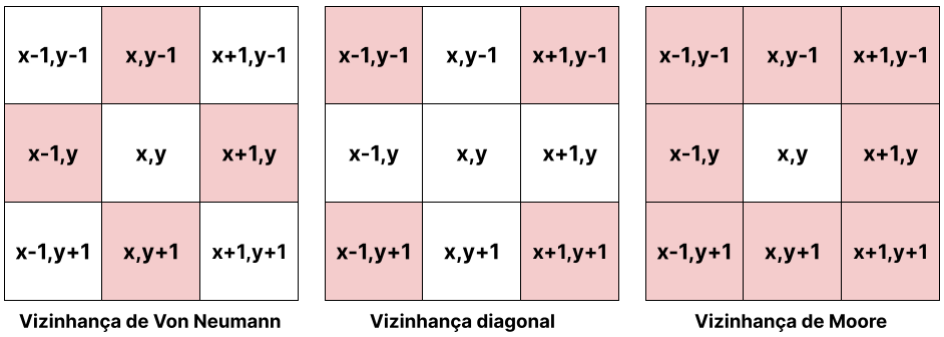

Figura 5.1 Ilustração dos tipos de vizinhança implementados, autor.

Antes de verificar se os vizinhos são acessíveis, é necessário verificar se se eles estão dentro dos limites do mapa:

```
def Verificar_Vizinhanca(x, y)
  Se x pertence ao mapa E y pertence ao mapa:
    retorne Verdadeiro
  Senão
    retorne Falso
```

A partir da implementação acima, basta percorrer cada célula do mapa e salvar os valores de quantidade de vizinhos vivo/mortos para cada célula visitada.

```
def Von_Neumann()
  mapa_vizinhos = []
  Para x no intervalo (0, mapa.largura)
    Para y no intervalo (0, mapa.altura)
      Se Verificar_Vizinhanca(x, y-1)
        mapa_vizinhos[x, y] = mapa[x, y-1] + mapa_vizinhos[x, y]
      Se Verificar_Vizinhanca(x-1, y)
        mapa_vizinhos[x, y] = mapa[x-1, y] + mapa_vizinhos[x, y]
      Se Verificar_Vizinhanca(x+1, y)
        mapa_vizinhos[x, y] = mapa[x+1, y] + mapa_vizinhos[x, y]
      Se Verificar_Vizinhanca(x, y+1)
        mapa_vizinhos[x, y] = mapa[x, y+1] + mapa_vizinhos[x, y]
```

De forma semelhante, a vizinhança diagonal funciona modificando as 4 condicionais do código acima

```
def Diagonal()
  ...
  Se Verificar_Vizinhanca(x-1, y-1)
    mapa_vizinhos[x, y] = mapa[x-1, y-1] + mapa_vizinhos[x, y]
  Se Verificar_Vizinhanca(x+1, y-1)
    mapa_vizinhos[x, y] = mapa[x+1, y-1] + mapa_vizinhos[x, y]
  Se Verificar_Vizinhanca(x+1, y-1)
    mapa_vizinhos[x, y] = mapa[x-1, y+1] + mapa_vizinhos[x, y]
  Se Verificar_Vizinhanca(x+1, y+1)
    mapa_vizinhos[x, y] = mapa[x+1, y+1] + mapa_vizinhos[x, y]
```

A vizinhança de Moore funciona combinando todas as condicionais vistas nas duas abordagens acima.

5.4.2 Estados do autômato

A sugestão de um ou mais estados extras foi satisfeita de forma a adicionar o estado de imutável a um CA. O estado imutável torna uma célula do autômato inalterável por quaisquer regras de vizinhança que tentem interagir. Normalmente, como descrito na seção de revisão literária, um autômato possui como motor a aplicação de uma regra em toda sua estrutura. Ao adicionar o

estado imutável, é esperado que o mapa preserve mais de suas características a cada iteração. Uma possível aplicação do estado imutável, por exemplo, é manter um layout pré-definido pelo usuário, ou ainda definir as bordas o mapa como imutáveis, de forma que as bordas possam ser sempre inacessíveis.

5.4.3 Modelo Contínuo

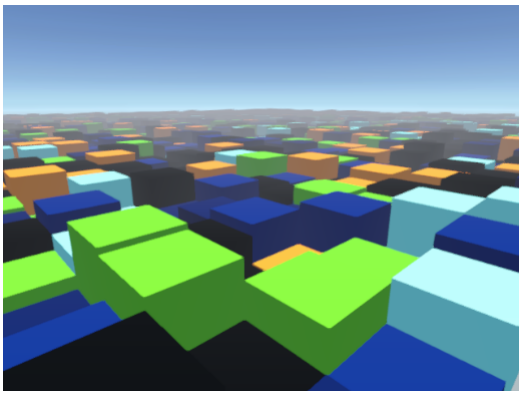

Figura 5.2 Captura de execução da ferramenta desenvolvida no Unity: mapa em 3 dimensões, autor.

Como sugerido: expandir o modelo de autômatos celulares para adicionar estados contínuos permite utilizar essa variável contínua como referencial de altura. Até então, cada célula do autômato precisaria ter uma altura definida por padrão e essa altura seria modificada por funções de erosão (pós-processadas). Através da implementação do atributo contínuo, torna-se possível gerar altura (ainda que aleatoriamente) durante a evolução do mapa, removendo a necessidade de realizar etapas posteriores de processamento. Normalmente, como mostrado nos trabalhos relacionados de [5], a utilização de algoritmos (Midpoint, Diamond, Worley, Perlin e Simplex) que podem gerar padrões de ruído de gradientes de cinza, permitem ao usuário simular texturas terrestres com grau de semelhança relativo a realidade. Entretanto, ao gerar a altura aleatoriamente utilizando como célula um autômato cúbico os resultados não foram são agradáveis (imagem acima). Acredito que a alternativa mais viável seria utilizar de funções aplicadas num plano dividido em quadrantes (os autômatos) para modelar o mesmo. Como alternativa a essa sugestão de modelo contínuo e das técnicas de erosão, uma abordagem que experimentei foi gerar (como sugerido) um CA em uma só camada, exportar esse mapa para uma imagem (100 células dispostas num *grid* 10x10, gerou uma imagem de 100 pixels, 10x10) e utilizar uma ferramenta de processamento de imagem externa (PIL, *Python Image Library*) para aumentar (técnica de *resampling* Laczos) a escala em 3 vezes (tamanho final 40x40) a do mapa original, resultando num modelo semelhante ao apresentado em [5] e que se assemelha a implementação contínua do Lênia.

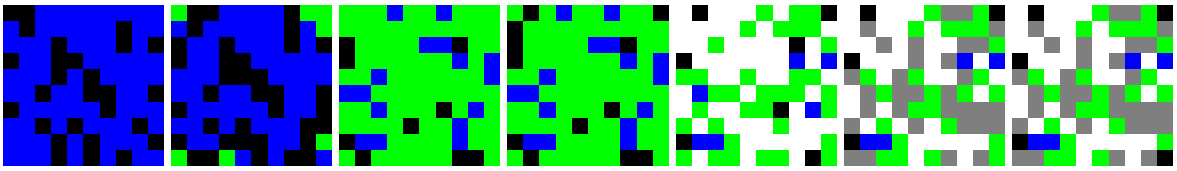

Figura 5.3 Evolução do autômato utilizando apenas uma camada, autor.

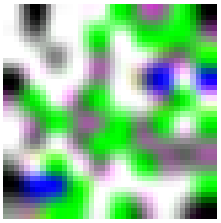

Figura 5.4 Escalonamento da imagem do último estado do autômato, autor.

Ao realizar o escalonamento, cores de tonalidade roxas/rosadas não presentes na montagem do automato surgem. Isso se dá por razão do algoritmo de escalonamento criar tons de transição entre um pixel e outro. Para normalizar as cores, pode-se utilizar da mesma técnica que algoritmos de padrões ruído citados acima fazem: representar a imagem em tons cinza.

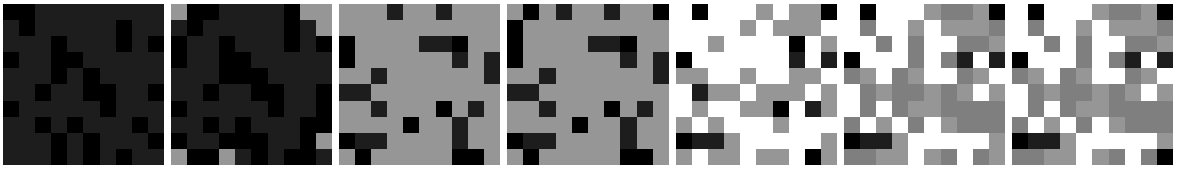

Figura 5.5 Evolução do autômato utilizando apenas uma camada em tons de cinza, autor.

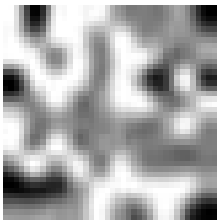

Figura 5.6 Escalonamento da imagem do último estado do autômato em tons de cinza, autor.

5.5 Experimentos e resultados

Nesta seção serão feitos experimentos e apresentados seus resultados com base na implementação dos trabalhos futuros do artigo de [5] e utilizando as mesmas métricas para comparação a partir de [2]. O objetivo é investigar se autômatos podem atingir a mesma qualidade na geração de mapas. Os experimentos foram realizados utilizando um *desktop* comum (processador

Intel Core i5 10400, 16 GB de RAM de frequência 2666 MHz, sem placa de vídeo (GPU), C# com compilador .NET versão 7.0.203, Python versão 3.10.4, no sistema operacional Windows 10, armazenamento em disco rígido de 7200 RPM). Para cada experimento foram salvos: um arquivo TXT contendo a cor da célula para cada iteração do autômato e um arquivo YAML contendo os registros da execução para serem processados na linguagem Python, os arquivos para reprodução e compilação própria estão disponíveis sob licença pública e hospedados no GitHub do autor.

Em [5], nas primeiras iterações de seu algoritmo são feitas num plano de 32x32 células, que utilizando da divisão celular e sucessivas aplicações posteriores de função de erosão, termina com um número maior. Isto posto, como não estarei utilizando da divisão celular nem de algoritmos de erosão, é tomada a decisão de utilizar um plano de 64x64 células, de forma a manter um nível de detalhamento intermediário. A partir de um experimento, busca-se atingir bons resultados para função fitness elucidada na seção de métricas para as propostas de trabalhos futuros identificadas na seção de implementação.

Isto posto, em todos os experimentos realizados foram utilizadas as seguintes variáveis padrão:

- Comprimento: 64 células
- Largura: 64 células
- Quantidade de células total: 4096
- Tipos distintos de células: 5 (acessíveis representadas pelas cores: azul, verde, branco, cinza e inacessível identificada pela cor preta)
- Evoluções do autômato realizadas: 5

A montagem do autômato em um plano é feito de forma aleatória, com seu estado de acessibilidade definido por uma função randômica de 50% de probabilidade. A cor (tipo) da célula é definido também de forma aleatória entre um dos 4 tipos de célula. As regras de evolução seguem as premissas dispostas na seção de revisão literária:

```
// Regra de reprodução
Se(célula está morta E quantidade de vizinhos == 3):
    próximo_estado( célula->viver() )


// Regras de subpopulação e superpopulação
Senão se(célula está viva E
        (quantidade de vizinhos < 2 OU quantidade de vizinhos > 4)):
    próximo_estado( célula->morrer() )

// Regra de equilíbrio
Senão:
    próximo_estado( célula )
```

5.5.1 Experimento 1

Neste primeiro experimento, foi testada a geração de um mapa balanceado, com proporção de células inacessíveis dividida igual às acessíveis. Foi estipulado, entretanto, um peso mais importante para células acessíveis (wa = 80%).

Tabela 5.1 Uma iteração de experimento com vizinhança Neumann.

Parâmetros	Valor
Pa	0.969238281
vt	1.25
vs	0.25
A	1.031738
Pe	3.0761719
E	32.50794
Et	0.32507935
Es	32.182858
we	0.2
wa	0.8
Tempo de execução	143ms
Fitness	6.6365714

O primeiro experimento não obteve resultados empolgantes. Apesar de utilizar o algoritmo de vizinhança de Neumann, apresentado em [5], para gerar o mapa, o valor de fitness excedeu em muito o esperado (esperava-se um número próximo de 0, tal qual exibido em [2]). Observando a ilustração do mapa resultante abaixo, é visível que ao utilizar apenas uma camada para abrigar tipos de autômato que interagem entre si acabou por deixar o mapa bastante poluído e esparso. Ao verificar o valor calculado de: w_a * v_s + w_e * E_s; percebe-se que o E_s está muito alto (32.18) e está afetando drasticamente o cálculo, essa constatação é reforçada pelo valor de P_a, que (0.96, equivalente a 96% ou 1-P_e). Isso acontece, pois como demonstrado na fórmula 5.8, o E_s é calculado a partir da quantidade de células inacessíveis; como existem poucas células inacessíveis, o valor dispara. Para tentar solucionar esse problema, no experimento 2 será utilizado um p_a baseado no p_{a2} (80%) e p_{e2} (25%) de [2], de forma a estabelecer um piso de células inacessíveis existentes, independente do que exija a função de vizinhança, o nome dessa vizinhança será notado como "Neumann Limitada".

```
// Modificação para Vizinhança de Neumann Limitada
// Regra de reprodução
Se(célula está morta E quantidade de vizinhos == 3
   OU quantidade células vivas < 20%):
    próximo_estado( célula->viver() )
```

```
// Regras de subpopulação e superpopulação
Senão se(célula está viva E
        (quantidade de vizinhos < 2 OU quantidade de vizinhos > 4)
        OU quantidade de células vivas > 80%):
    próximo_estado( célula->morrer() )

// Regra de equilíbrio
Senão:
    próximo_estado( célula )
```

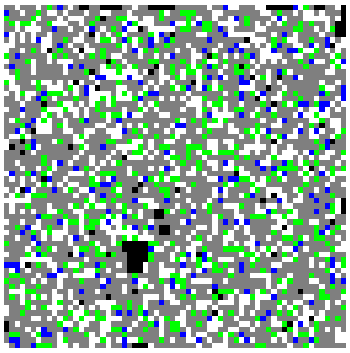

Figura 5.7 Mapa gerado no experimento número 1, autor.

5.5.2 Experimento 2

No experimento 2 será alterada a regra de vizinhança baseada no Game of Life. Percebe-se que é necessário haver garantia de proporção e maior quantidade de células inacessíveis, de forma a reduzir o caos do sistema. Ao observar os parâmetros utilizados por [2], percebe-se que ao atribuir valores máximos de P_a como 80% e P_e 25% bons resultados foram atingidos, por isso, esse serão os intervalos utilizados no experimento abaixo.

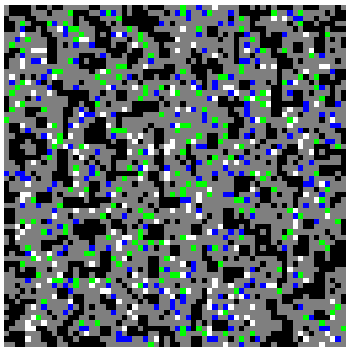

Figura 5.8 Mapa gerado no experimento 2, autor.

Tabela 5.2 Uma iteração de experimento com vizinhança Neumann Limitada

Parâmetro	**Valor**
Pa	0.72192383
vt	1.25
vs	0.25
A	1.3851877
Pe	27.807617
E	3.596137
Et	0.03596137
Es	3.5601757
we	0.2
wa	0.8
Tempo de execução	153ms
Fitness	0.9120351

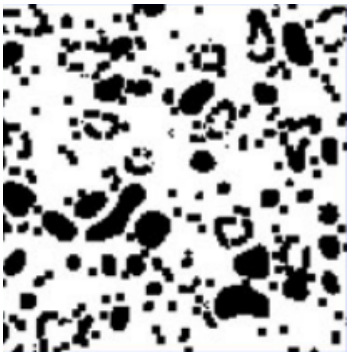

Figura 5.9 Figura 19 retirada de *Automatic evolution of programs for procedural generation of terrains for video games: Accessibility and edge length constraints*, [2].

O fitness foi melhorado drasticamente após a modificação das regras de vizinhança. A adição de uma condicional "e" para restringir as modificações da função de transição impactaram na redução de mais de 6 vezes o valor do experimento 1. A imagem agora se assemelha um pouco mais semelhante a casos como a figura 19 da pesquisa de [2], que possui fitness perfeito (0.0), ilustrada abaixo na direita.

Antes de avaliar outros melhoramentos, é necessário verificar se o valor do experimento 2 foi um valor aberrante (*outlier*). Foram obtidos os seguintes resultados:

Tabela 5.3 Dez mil iterações com método de vizinhança Neumann Limitada

Experimento	**Resultados de *Fitness***
Média	1.0064278238870001
Desvio padrão	0.009541613076023255
Menor valor	0.99399
25% (1o quartil)	0.99919134
50% (2o quartil)	1.0042702
75% (3o quartil)	1.0112238
Maior valor	1.070348
Tempo de execução médio	158.6432 ms

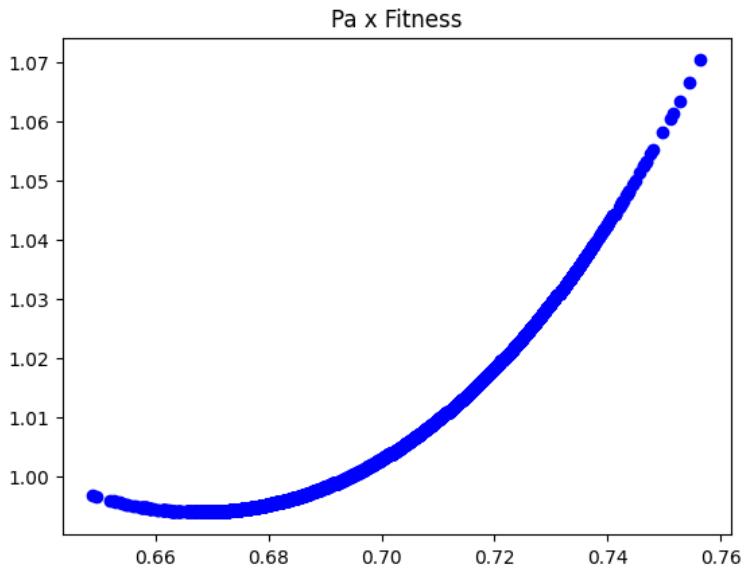

Figura 5.10 Gráfico do experimento 2 de P_a x *Fitness* resultante das dez mil execuções, autor.

A partir da execução de dez mil autômatos gerados utilizando os mesmos parâmetros do experimento 2, podemos identificar que o P_a apresenta um comportamento semelhante a uma função quadrática com concavidade para cima. Este tipo de função possuí um mínimo global, e, conforme observados os valores de desvio padrão, ele possivelmente foi atingido nos menores intervalos. Na imagem é possível perceber que entre os valores de P_a (eixo X) 0.66 e 0.68 de P_a há um melhor rendimento de fitness, ao passo em que valores anteriores e posteriores a esse trecho ocorre piora do fitness. Isso se confirma quando comparamos com o que foi visto no experimento 1: havia um P_a menor e o fitness foi muito mais alto. Além da variável P_a, os outros parâmetros atingiram baixo valor de desvio padrão, sendo apenas os valores de células acessíveis e inacessível que se modificaram mais (como esperado). O tempo de execução médio foi de 158ms, com desvio padrão de 122ms, e valor de mínimo de 116ms e máximo de 2427ms. Dado que foram, a partir de [2] e de nossos experimentos, foram exauridas as capacidades de interpolação de parâmetros, podemos alternar para outros tipos de vizinhança implementadas.

A próxima distância a ser experimentada, ainda que semelhante a de Neumann, é a diagonal.

Por ter quatro células em sua vizinhança, é esperado que a função de fitness apresente resultados condizentes com os dos experimentos passados. O que há de ser observado é se o visual do mapa ficará mais disperso ou mais coeso que os observados nas vizinhanças de Neumann e Neumann Limitada.

5.5.3 Experimento 3

Tabela 5.4 Dez mil iterações com método de vizinhança Diagonal

Métrica	**Resultados de *Fitness***	**Resultados de P_a**
Média	0.783917925098	0.388840136638
Desvio padrão	0.10812665604187927	0.02206848306405352
Menor valor	0.29754987	0.29882812
25% (1o quartil)	0.71218383	0.37402344
50% (2o quartil)	0.7810138	0.38867188
75 (3o quartil)	0.8539374	0.40356445
Maior valor	1.0959013	0.46655273
Tempo de execução médio	174.7844 ms	

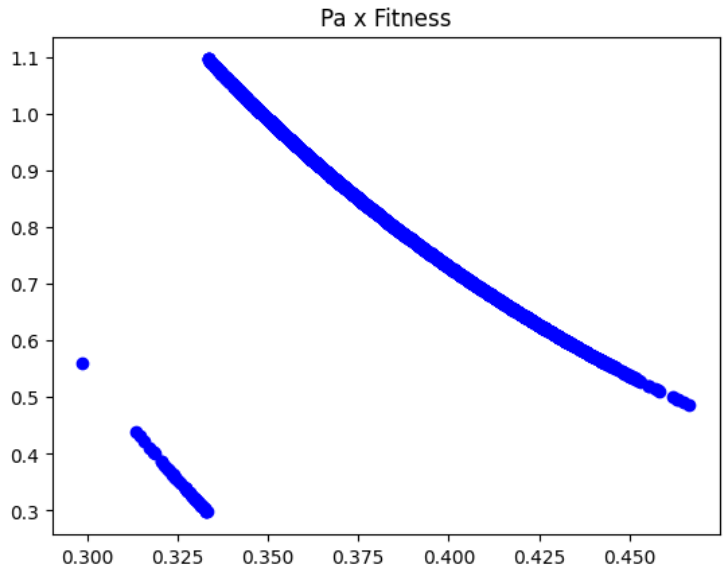

Figura 5.11 Gráfico do experimento 3 de P_a x *Fitness* resultante das dez mil execuções, autor.

Pode-se dizer que o terceiro experimento atinge um recorde, tendo seu melhor *fitness* 0.29 na iteração 1101, com P_a de 0.29. Isso significa que menores P_as resultaram em *fitness* maiores para esta configuração. Entretanto, ao observar o gráfico que compara P_a com *fitness* é

possível ver que os resultados não apresentam uma continuidade, sendo o melhor *fitness* atingido claramente um *outlier*. Abaixo, observando a imagem a esquerda, que ilustra o melhor caso alcançado, é perceptível que, ao proliferar várias áreas inacessíveis o mapa adquire uma característica labiríntica. Como exposto na seção de revisão literária, sabe-se que autômatos podem ser utilizado para gerar mapas úteis para jogos de RPG (com destaque para *dungeons*), e esse *fitness* beneficiou a geração deste cenário. Na imagem da direita, a qual foi retirada de um resultado com *fitness* de 0.46453974 e que segue a curva da função quadrática observada acima, é perceptível que existem mais terrenos acessíveis, ainda assim, mapa resultante é bastante acidentado. Embora estes resultado possam não ser o melhor uso para jogos RTS, onde são desejados espaços largos para deslocamento de unidades e posicionamento de estruturas, é possível exportá-los e adaptar o mapa ao aplicar-se pós processamento e erosão, técnicas descritas em [5].

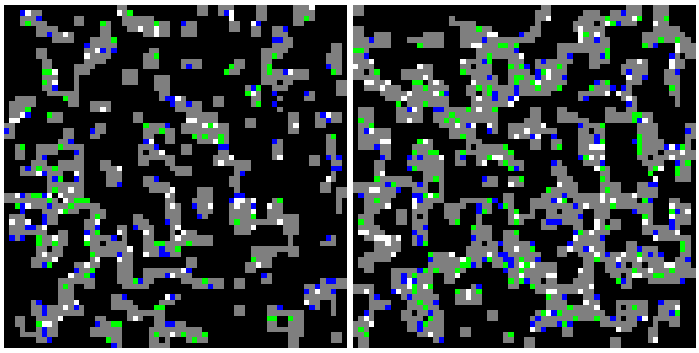

Figura 5.12 À esquerda: gráfico do melhor *fitness* do experimento 3; À direita gráfico de um fitness pertercente a função quadrática observada no experimento 3, autor.

5.5.4 Experimento 4

Tabela 5.5 Dez mil iterações com método de vizinhança Diagonal Limitada

Métrica	**Resultados de *Fitness***	**Resultados de P_a**
Média	1.0025440435070003	0.695862671141
Desvio padrão	0.007489720421042951	0.014063042764821228
Menor valor	0.99399	0.6442871
25% (1o quartil)	0.99684083	0.6862793
50% (2o quartil)	1.0007632	0.6960449
75% (3o quartil)	1.0061538	0.70532227
Maior valor	1.0595226	0.7504883
Tempo médio de execução	181.6359 ms	

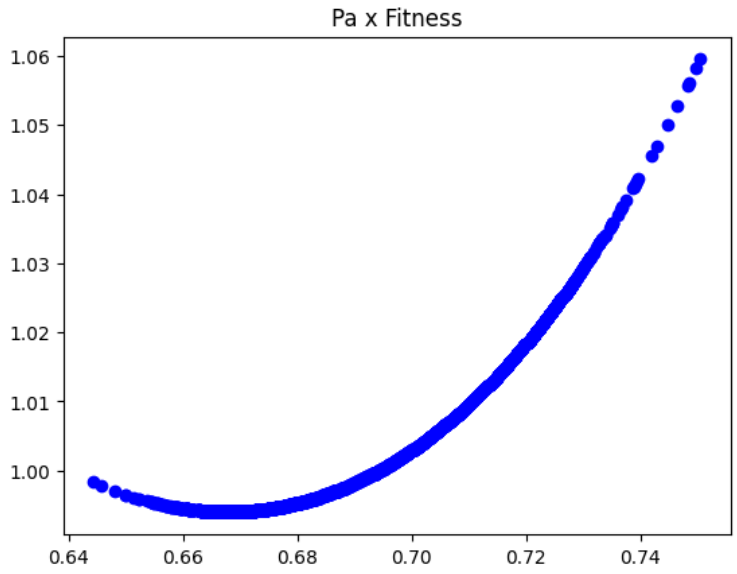

Figura 5.13 Gráfico do experimento 4 de P_a x *Fitness* resultante das dez mil execuções, autor

Ao tentar controlar o experimento 3 utilizando da modificação também aplicada ao experimento 2 (limitar proporção de células vivas/mortas), ocorre o negativo efeito de aumentar o valor de *fitness*. Essa abordagem provou-se infrutífera no experimento 4. No próximo experimento, que avalia uma vizinhança maior superior a 4 células, será reutilizada essa mesma abordagem, dado que as regras descritas em Game of Life não são adaptadas para esse cenário.

5.5.5 Experimento 5

A vizinhança de Moore possui uma característica particular em relação as anteriores: ela verifica uma vizinhança de 8 células adjacentes, sendo o dobro de Neumann e das diagonais. Ao executar os testes utilizando as mesmas regras apresentadas em Game of Life, o algoritmo excede a quantidade de células mortas (regra de superpopulação). Isso é bastante claro, dado que, ao visitar o dobro de células, a proporção definida por Conway não se beneficiaria da regra desta vizinhança. Não foram encontrados resultados competitivos utilizando função de controle do semelhante a do experimento passado, com *fitness* médio ficando em torno de 1.03. Além disso, uma das vantagens de se estar utilizando autômatos foi colocada em risco: ao realizar mais verificações condicionais para a vizinhança, o tempo médio de execução subiu para 395ms, um incremento de 1 vezes quando comparada aos experimentos passados (variam em torno de 150 a 180ms).

5.5.6 Experimento 6

Para testar a implementação proposta do estado adicional imutável (célula que, uma vez viva/morta atribuída, não muda de estado), foram executadas 900 iterações para cada método de vizinhança. A cada iteração o número de autômatos imutável subia 0.1%, começando de 0% até 90%. Não foram utilizadas regras extras para garantir a quantidade proporcional, visto que

essa função foi infrutífera nos casos anteriores.

Tabela 5.6 Tabela de desvio padrão, valores mínimo e máximo, quartis e tempo médio de execução para as abordagens Neumann, Diagonal e Moore.

Abordagem	**Neumann**	**Diagonal**	**Moore**
Fitness Médio	0.662434	0.657863	0.596421
Desvio Padrão	0.336423	0.327487	0.320405
Menor Valor de Fitness	0.396000	0.396000	0.396000
1º quartil	0.414686	0.415305	0.410386
2º quartil	0.450511	0.457641	0.424064
3º quartil	1.168999	1.164815	0.486971
Maior Valor de Fitness	1.195413	1.195413	1.195413
Tempo Médio de Execução	604.843333	595.218889	600.808889

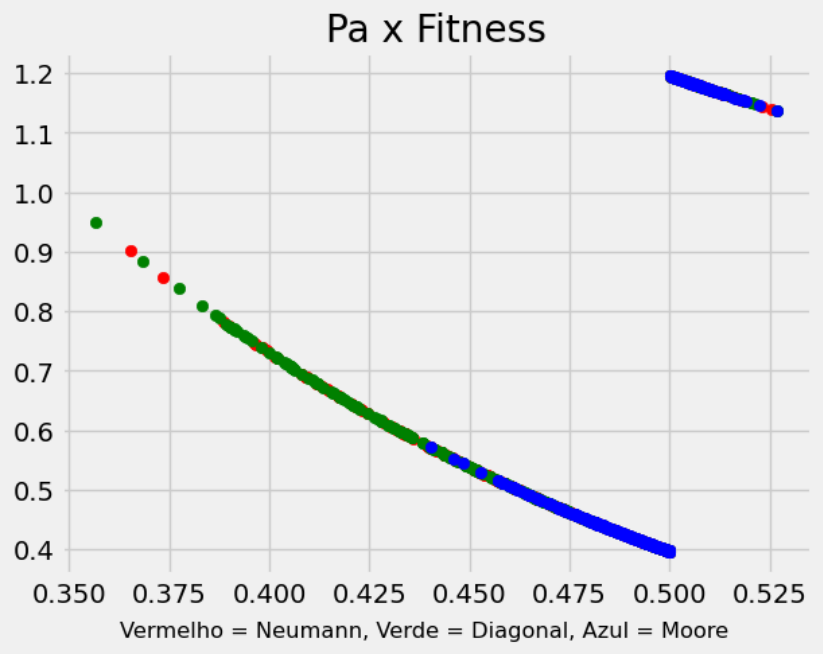

Figura 5.14 Gráfico agregado combinando dados das técnicas de Neumann, Diagonal e Moore, autor.

O gráfico exibido acima demonstra a constância entre os 3 algoritmos testados: quando sobrepostos, são poucas as diferenças entre os valores analisados. O algoritmo de Moore, entretanto, é a melhor das alternativas, com aproximadamente 10% de vantagem. Quanto a Neumann e Diagonal, são virtualmente iguais, com uma leve vantagem para o método de vizinhança Diagonal por ser 2% mais rápido.

5.6 Conclusão e trabalhos futuros

Os experimentos apresentados na seção passada demonstram o uso e aplicação de autômatos celulares na criação de arte (em formato de mapas) através de criatividade computacional. Ao combinar conteúdo de dois artigos visando utilizar das mesmas métricas para competir com algoritmos genéticos, usando o potencial emergente dos autômatos celulares, não foi possível atingir resultados tão próximos. Isso ocorreu, pois a abordagem de algoritmos genéticos garante a busca implacável por um resultado refinado que é norteado pela sua função de *fitness*. Ainda assim, a abordagem caótica e estocástica dos autômatos celulares pode servir para aplicações na área dos jogos. Ela consome, sem dúvidas, menos recursos computacionais por envolver menos cálculos, e é interessante caso seja necessário aplicar em dispositivos com recursos limitados (por memória, bateria, processamento, etc). Entretanto, se há capacidade e necessidade de atingir os melhores resultados possíveis, a alternativa de algoritmos genéticos seria a indicação.

Uma possibilidade futura a ser explorada é utilizar do vasto catálogo da enciclopédia de espécies do Game of Life para experimentar modelos descobertos e com comportamento previsto. Podem existir aplicações práticas que se beneficiem do uso de autômatos regulares, como controladores de luz ou outros dispositivos com comportamento periódico.

Após avaliar exaustivamente a caoticidade dos mapas gerados por autômatos em uma só camada, outra boa possível boa aplicação de autômatos celulares ou estudo de caso de uso seria aplicação em jogos do tipo *match-3*, como Candy Crush, Tetris ou Bejeweled, ou jogos de roleta de cassino. Esses jogos podem abusar dos pontos positivos de autômato: alta aleatoriedade e baixo custo de processamento. O estudo de caso seria voltado a aplicações de autômatos nesse contexto: probabilidade de células adjacentes combinarem, grau de entropia de um mapa e elaboração de funções de transição com regras de ativação com viés para preservar a entropia.

Referências Bibliográficas

[1] Google, "Liquidfun," 2014. [Online; Arquivo acessado em Fevereiro/2023.].

[2] d. V. F. . C. Frade, M., "C. automatic evolution of programs for procedural generation of terrains for video games.," *Soft Comput 16*, p. 1893–1914, 2012.

[3] E. W. WEISSTEIN, "A new kind of science." Also published on Wolfram MathWorld, escrito em 2002, acessado em Dezembro/2022.

[4] S. ARBESMAN, "Emergent microcosms: Artificial life, agent-based modeling, and unspooling worlds with code.." Published on Substack blog post 13/12/2022, acessado em Dezembro/2022.

[5] P. Ziegler and S. von Mammen, "Generating real-time strategy heightmaps using cellular automata," in *Proceedings of the 15th International Conference on the Foundations of Digital Games*, FDG '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[6] B. K. P. Horn, "Hill shading and the reflectance map," *Proceedigs of the IEEE, 69(1)*, pp. 14–47, 1981.

[7] M. GARDNER, "The fantastic combinations of john conway's new solitaire game 'life'," *Scientific American 223*, pp. 120–123, 10 1970.

[8] Wikipédia, "Geração procedural em jogos digitais — wikipédia, a enciclopédia livre," 2021. [Online; accessed 18-novembro-2021].

[9] B. W.-C. CHAN, "Lenia: Biology of artificial life," *Complex Systems*, vol. 28, pp. 251–286, 2019.

[10] M. BITTKER, "Sandspiel." Publicado em 19/04/2019, acessado em Março/2023.

[11] T. ÖSTERLUND, "Design possibilities of emergent algorithms for adaptive lighting system," *Nordic Journal of Architectural Research*, vol. 25, pp. 159–184, 11 2013.

[12] "Silo 468 / lighting design collective," 2012. Arquivo acessado em Fevereiro/2023.

[13] C. INSOOK, "Interactive sonification exploring emergent behavior applying models for biological information and listening," *Frontiers in Neuroscience*, vol. 12, 2018.

[14] I. Wolfram Research, “Wolframtones: How it works,” 2005. [Arquivo acessado em Março/2023.].

[15] W. Alexan, M. ElBeltagy, and A. Aboshousha, “Rgb image encryption through cellular automata, s-box and the lorenz system,” *Symmetry*, vol. 14, no. 3, 2022.

[16] D. H. I. K. N. M. Husnul Habib Yahya, H. Fabroyir and S. Arifiani, “Dungeon’s room generation using cellular automata and poisson disk sampling in roguelike game,” *13th International Conference on Information & Communication Technology and System (ICTS)*, pp. 29–34, 2021.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).