UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE ARTES E LETRAS
CURSO DE ESPECIALIZAÇÃO EM DESIGN DE SUPERFÍCIE

Felipe Paetzhold Körbes

PROCESSOS GENERATIVOS DE IMAGEM COMO BASE PARA A CRIAÇÃO DE DESIGN DE SUPERFÍCIE

Santa Maria, RS, Brasil

2019

Felipe Paetzhold Körbes

PROCESSOS GENERATIVOS DE IMAGEM COMO BASE PARA A CRIAÇÃO DE DESIGN DE SUPERFÍCIE

Monografia apresentada ao Curso de Especialização em Design de Superfície, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de Especialista em Design de Superfície.

Orientador: Wagner de Souza Antonio

Santa Maria, RS, Brasil

2019

FICHA CATALOGRÁFICA

KÖRBES, Felipe Paetzhold, 1992 -

Processos Generativos de Imagem como Base para Criação de Design de Superfície – Santa Maria, RS: Curso de Especialização em Design de Superfície/

Universidade Federal de Santa Maria, RS/ Felipe Paetzhold Körbes, 2019.

128p.

Orientador: Wagner de Souza Antonio.

Monografia (especialização) – Universidade Federal de Santa Maria, Centro de Artes e Letras, Curso de Especialização em Design de Superfície, RS, 2019.

1. Design 2. Design de Superfície 3. Design Generativo 4. Programação Criativa

Endereço: Travessa Antônio Baggio, n. 89, Ijuí, RS. CEP: 98700-000

Fone (0xx)55 984496859; E-mail: felipe.korbes@gmail.com

Felipe Paetzhold Körbes

PROCESSOS GENERATIVOS DE IMAGEM COMO BASE PARA A CRIAÇÃO DE DESIGN DE SUPERFÍCIE

Monografia apresentada ao Curso de Especialização em Design de Superfície, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de Especialista em Design de Superfície.

Aprovado em 6 de Dezembro de 2019.

_____

Wagner de Souza Antonio, Ms.

(Presidente/Orientador)

_____

Carolina Iuva de Mello, Dr.ª

_____

Reinilda de Fátima Berguenmayer Minuzzi, Dr.ª

Santa Maria, RS, Brasil

2019

AGRADECIMENTOS

Gostaria de agradecer aos meus pais, José e Anísia pelo apoio, moral e financeiro, pelo incentivo e pela paciência ao longo do curso e do trabalho.

Agradeço ao professores do curso pelo conhecimento que me transmitiram. Agradeço especialmente ao professor e orientador Calixto Bento pela oportunidade e disponibilidade. Agradeço também às professoras Reinilda e Carol, pelo interesse e pela participação na banca.

RESUMO

PROCESSOS GENERATIVOS DE IMAGEM COMO BASE PARA A CRIAÇÃO DE DESIGN DE SUPERFÍCIE

AUTOR: Felipe Paetzhold Körbes

ORIENTADOR: Wagner de Souza Antonio

O presente projeto monográfico consiste em uma pesquisa em design de superfície com foco no uso de Sistemas Generativos de Imagem. A partir de uma investigação teórica sobre alguns conceitos base no campo do sesign de superfície, assim como conceitos relacionados ao design de sistemas generativos de imagem, esta pesquisa de caráter prático, exploratório e aplicado visa relacionar as possibilidades do uso de sistemas generativos com a prática do design de superfície, de modo a ampliar as possibilidades e recursos à disposição do designer. Para este fim, além da pesquisa teórica em bibliografias no campo do design de superfície, das artes e conceitos relacionados ao uso de sistemas generativos de imagem, foi feito um estudo de caso, com o relato do desenvolvimento de um conjunto de sistemas generativos para fins de design de superfície. Os sistemas generativos foram desenvolvidos à base de programação na linguagem Processing. O produto do processo de desenvolvimento é um conjunto de programas generativos com movimento e comportamentos baseados em comportamentos de bando encontrados na natureza que podem ser aplicados através de projeção digital sobre superfície. Os programas também possuem a capacidade de salvar imagens do estado visual de cada programa para uso em padrões visuais tradicionais. Este documento contém, então, seções relacionadas ao design generativo, ao design de superfície e o relato do processo criativo de sistemas generativos para uso em design de superfície, assim como os resultados alcançados.

Palavras-chave: Design, Design de Superfície, Design Generativo, Programação Criativa.

ABSTRACT

GENERATIVE PROCESSES OF IMAGES AS BASIS FOR SURFACE DESIGN

Author: Felipe Paetzhold Körbes

Supervisor: Wagner de Souza Antonio

The current monographic project consists in a surface design research, with focus on the use of Generative Image Systems. From a theoretical investigation on a few key concepts in the field of surface design, as well as concepts related to generative image systems design, this research, which has a practical, exploratory and applied bias, aims to relate the possibilities of the use of generative image systems with the practice of surface design, in a way to expand the repertoire of possibilities and resources available to the designer. To this end, beyond the theoretical research in surface design, arts and generative systems bibliographies, a case study was made, with an account of the development of a set of generative surface design systems. The generative systems were developed on the Processing programming language. The product of the development process is a set of generative applications with movement and behavior based on pack behaviors found in nature that can be applied through digital surface projections. The applications also have the capacity to save images of the visual state of each application to be used in traditional visual patterns. This document therefore contains sections related to generative design, surface design,and the account of the creative process of the development of generative surface desigin systems, as well as the results achieved.

Key-words: Design, Surface Design, Generative Design, Creative Coding.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

A tecnologia digital tem se tornado cada vez mais parte importante nas práticas de design. Com o constante desenvolvimento de ferramentas e tecnologias voltadas para o desenvolvimento de produtos de design as possibilidades de expressão criativa se expandem rapidamente. Considerando esse crescimento tecnológico constante se faz necessária a constante adaptação dos profissionais de design às características e possibilidades que se desdobram deste novo ambiente de desenvolvimento criativo.

Apesar de pervasiva no cotidiano e usada para desenvolvimento científico de forma constante, a tecnologia digital e suas linguagens ainda são limitadas em seu uso ao que se refere ao design e a expressão visual criativa de modo geral, frequentemente se limitando a simulações de ferramentas analógicas, que apesar de possibilitar processos de criação extremamente eficientes e inclusive linguagens próprias, limitam profundamente as possibilidades reais do que as capacidades de processamento de dados podem oferecer. Cabe então explorar estas possibilidades, que são únicas ao meio digital e aprender os métodos, ferramentas e ambientes de desenvolvimento generativo de imagens para desenvolver novas linguagens e expandir os limites da expressão visual.

Nos voltamos, portanto, para o design generativo*,* um prática de produção visual que apesar de não ser limitada ao ambiente digital, por consequência da necessidade de definições de instruções e iterações, utiliza capacidades da tecnologia digital de forma diferente de outras formas de produção.

1.1. OBJETIVOS

1.1.1. Objetivo Geral

Estabelecer um conhecimento base sobre processos generativos de criação para desenvolver um conjunto de sistemas generativos de imagem para uso em superfícies.

1.1.2. Objetivos Específicos

- Explorar o design de superfície e apresentar algumas das possibilidades criadas pelo uso de novas técnicas e tecnologias na área;
- Pesquisar as possibilidades e aplicações atuais do design generativo e a associação destas com o design de superfície;
- Entender os conceitos normalmente associados aos processos generativos, como algoritmo, emergência e sistemas complexos.
- Analisar comportamentos naturais de bando para inspirar o desenvolvimento

dos sistemas generativos

1.2. JUSTIFICATIVA

É possível perceber um interesse em integrar estética e procedimentos baseados em métodos lógicos em diversas culturas ao longo do tempo, influenciando a cultura material de forma significativa. Os objetos criados criados com base neste interesse, apesar de conter padrões algorítmicos complexos baseados em princípios matemáticos, foram em sua maioria criados por artesãos, e não matemáticos. Isto indica a existência de um impulso natural de conciliar e combinar noções qualitativas de estética a sistemas técnicos e lógicos para a estruturação de resultados visuais.

Sistemas generativos são relevantes para o desenvolvimento do design de diversas formas. A integração destes sistemas aos processos de design possibilita o desenvolvimento e a exploração de novas soluções de design, impossíveis de outras formas. Além disso, como McCormack, Dorin e Innocent (2004) indicam, os processos gerativos podem, por consequência de seus diferentes princípios filosóficos e operacionais, acabar influenciando o campo do design como um todo, trazendo ideias de evolução, adaptação, herança de características e significados, entre outros.

Ainda de acordo com McCormack, Dorin e Innocent (2004) as principais características associadas aos sistemas generativos são:

- A habilidade de gerar complexidade estrutural ou comportamental através da combinação e associação de componentes de complexidade menor;
- O complexo relacionamento entre unidades individuais e o ambiente onde essas se encontram;
- A capacidade de manter-se relevante através das constantes mudanças e adaptações geradas pelo sistema;
- Capacidade de gerar estruturas, comportamentos, relacionamentos e resultados inéditos de forma autônoma.

Considerando a possibilidade de associar os métodos generativos de imagem ao design de superfície, identifica-se uma oportunidade de explorar os efeitos e resultados do uso de sistemas generativos como base de desenvolvimento de design, extrapolando os métodos considerados mais tradicionais dentro do design de superfície. Existe também a possibilidade de verificar a função do designer frente a um processo de design alternativo, onde o papel do designer passa a ser ao mesmo tempo o de um agente criativo e de um agente espectador e curador dos resultados vindos de um sistema autônomo.

1.3. ESTRUTURA DO TRABALHO

Para desenvolver o presente trabalho foi utilizada a revisão bibliográfica de conteúdo pertinente. A pesquisa bibliográfica focou-se na área do design de superfície, sistemas generativos e no design para uma compreensão básica do campo, e na associação do design com sistemas generativos. Após a revisão bibliográfica, esta foi aplicada em um estudo exploratório e experimental, na forma do desenvolvimento de uma série de sistemas generativos de imagem para uso em superfícies, utilizando bibliografias no design de superfície, nas artes, no design gráfico e com as metodologias descritas por Bonsiepe (1984), Baxter (2017) e Valério(2013) como base.

Este documento monográfico se estrutura em quatro capítulos. Após o capítulo que introduz as premissas relacionadas ao trabalho, no segundo capítulo são apresentadas alguns conceitos pertinentes, com fontes bibliográficas relacionadas ao campo do design de superfície. No terceiro capítulo é abordado o design e os sistemas generativos, algumas de suas definições, assim como suas relações com conceitos tangentes que possam se apresentar relevantes para sustentar a exploração e o desenvolvimento que se dá no capítulo seguinte. O capítulo quarto apresenta o método e o processo de desenvolvimento de um conjunto de sistemas generativos programados por código e a aplicação destes. O trabalho conclui com a apresentação dos resultados obtidos.

2. DESIGN DE SUPERFÍCIE

Para explorar as possibilidades dentro do design de superfície, se faz necessário trazer um conceito preciso para a área, e delimitar as práticas que se encontram pertencentes à prática do design de superfície. Rüthschilling (2008) traz a seguinte definição para a atividade:

> Design de Superfície é uma atividade criativa e técnica que se ocupa com a criação de desenvolvimento de qualidades estéticas, funcionais e estruturais, projetadas especificamente para constituição e/ou tratamentos de superfícies, adequadas ao contexto sociocultural e às diferentes necessidades e processos produtivos. (RÜTHSCHILLING,2008).

Ainda considerando o design de superfície, Rubim (2004) indica que pela sua própria natureza, a atividade lida particularmente com considerações de ordem estética.

Tradicionalmente, os meios onde a prática do design de superfície é aplicada incluem o design para papel, incluindo estampas para papeis de parede e embrulho, o design voltado às superfícies têxteis, tomando a forma de estamparia, tecelagem, malharia, tapeçaria e outros, e o design de revestimentos cerâmicos, tanto nos aspectos bidimensionais quanto nas texturas tridimensionais destes. Considerando porém o constante desenvolvimento tecnológico, novas aplicações e possibilidades constantemente se emendam ao que tipicamente se entende por “Design de Superfície”.

Apesar de frequentemente apresentar as funções de tratamento e revestimento, como uma “pele” ou camada externa dos produtos, o design de superfície pode englobar o próprio produto. A definição de Schwartz (2008) amplia o conceito de design de superfície:

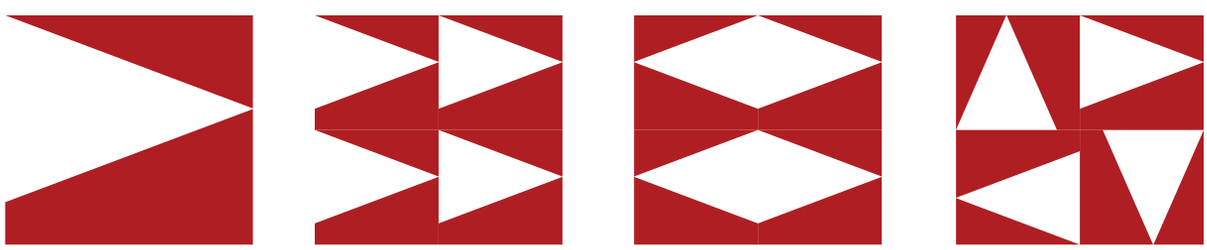
> Design de superfície é uma atividade projetual que atribui características perceptivas expressivas à Superfície dos objetos, concretas ou virtuais, pela configuração de sua aparência, principalmente por meio de texturas visuais, táteis e relevos, com o objetivo de reforçar ou minimizar as interações sensório-cognitivas entre o objeto e o sujeito. Tais características devem estar relacionadas às características, aspectos, funções estéticas, simbólicas e práticas (funcionais e estruturais) dos artefatos das quais fazem parte, podendo ser resultantes tanto da configuração de objetos pré existentes em sua camada superficial quanto do desenvolvimento de novos objetos a partir da estruturação de sua superfície. (SCHWARTZ, 2008).

Ruthschilling(2008) indica ainda que não existe uma fórmula estática a ser seguida, e que, por ser um descendente da prática da arte, o design de superfície adquire a liberdade de criação que vem com o domínio da linguagem visual e de lógicas criativas autorais.

2.1. ELEMENTOS BASE DO DESIGN DE SUPERFÍCIE

Considerando a abordagem tradicional do design de superfície, um dos elementos básicos associados à atividade é o módulo. Rüthschilling(2008) define o módulo como “a menor área que inclui todos elementos visuais que constituem o desenho”. Lupton e Phillips(2015) indicam que o módulo se caracteriza por ser um elemento fixo utilizado no contexto de um sistema ou estrutura maior. Estes módulos são frequentemente utilizados em sistemas de repetição, principalmente quando projetados para uso em superfícies contínuas, como tecidos a metro. Esta repetição tradicionalmente se dá através da translação, rotação e reflexão ordenada do módulo. Na figura 1 podemos visualizar um módulo sofrendo as operações mencionadas. Um termo comumente dado ao módulo em repetição no design de superfície, quando este possui um desenho com encaixe perfeito quando disposto, é *rapport (RUTHSCHILLING, 2008).* O produto da repetição ordenada de um módulo é chamado por fim de padrão, ou padronagem.

Figura 1. Módulo e repetição.



Fonte: Elaborado pelo autor, 2019.

Rubim (2004) frisa a importância de uma boa aplicação e exploração dos módulos para a construção de um padrão bem-sucedido. Ela indica ainda que a complexidade ou a “composição interessante” pode ser obtida da repetição em si, mesmo quando o módulo é simples ou básico.

> [...] Uma das coisas mais importantes na área é aprender como criar e projetar um desenho pois, uma imagem relativamente simples pode se tornar uma composição interessante e cativante, em virtude de ter sido habilmente transformada numa padronagem cujo desenho básico está em repetição[...]. (RUBIM, 2004)

Apesar do design de superfície não se limitar ao desenvolvimento de padrões, este continua sendo um elemento importante para a compreensão do campo e possui a característica de possibilitar ao designer a introdução de complexidade ao que num primeiro momento pode parecer ser um módulo ou desenho simplório.

2.2. NOVAS TÉCNICAS E TECNOLOGIAS NO DESIGN DE SUPERFÍCIE

Com o desenvolvimento de novas técnicas e tecnologias de impressão,

produção e tratamento de superfícies, novas práticas se abrem para utilização no projeto. O padrão que antes era limitado a um número fixo de módulos e a um conjunto fixo de cores por consequência das limitações técnicas e tecnológicas de produção, passa a apresentar novas possibilidades a serem exploradas, como um padrão têxtil contínuo onde o computador pode alterar a disposição de um módulo a ponto que o tecido não se repita em sua continuidade, como no exemplo da figura 2, onde um padrão têxtil é gerado digitalmente ao longo da impressão, alternando aleatoriamente a orientação rotacional de um módulo simples, e que, apesar de apresentar um encaixe visual, não se utiliza do *rapport* tradicional, fazendo com que o tecido tenha uma continuidade não repetitiva.

Figura 2. Padrão criado por Casey e Cait Reas, onde o computador decide a direção de um módulo simples, sem uso de *rapport, à direita exemplo de como o módulo é operado*.

Fonte: Casey Reas, Adafruit e Autor, 2012, 2014 e 2019.

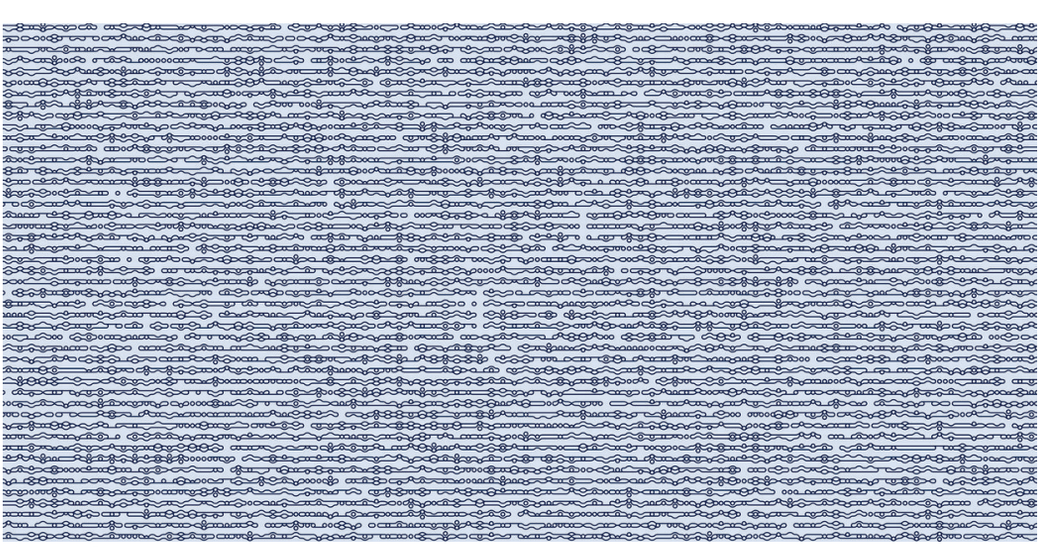
Outra possibilidade recente que surgiu da popularização e evolução das capacidades de projeção de imagens sobre superfícies é o uso de imagens digitais aplicadas diretamente sobre objetos, através de projetores de luz. Esta técnica, também conhecida como *video* ou *projection mapping,* ou mapeamento de projeção, permite o uso de imagens digitais em tempo real sobre superfícies que possam apresentar complicações — legais, físicas, técnicas ou logísticas — para o tratamento de sua superfície de formas tradicionais (DAY, 2012). O projeto 555 KUBIK (figura 3) utilizou-se da técnica do mapeamento de projeções para desenhar sobre um prédio com padrões e animações. Partindo do meio do mapeamento de projeções, e considerando a popularização e disponibilidade de telas digitais, estas também passam a fazer parte dos círculos onde o design de superfície atua, sob um diferente paradigma criativo, com outras limitações e outras possibilidades criativas.

Figura 3. 555 KUBIK, Projeto de projeção sobre arquitetura.

Fonte: Urban Screen, 2019.

É importante então, compreender que as possibilidades alcançadas recentemente através da evolução técnica da área permite o estudo e desenvolvimento de projetos que não estão restritos às técnicas consideradas tradicionais na área, como padronagem ou o *rapport.* Um exemplo pode ser visto na figura 4, onde, de forma similar ao exemplo da figura 2, um programa foi criado para gerar um desenho para impressão digital sobre tecido ao longo do processo de impressão, onde os elementos não se repetem de maneira ordinária, porém mantém uma unidade visual estética.

Figura 4. Pano pra manga, um padrão sem repetição, criado por programação para impressão digital.



Fonte: Contrast.parts, 2018.

O uso de novos processos e técnicas no design de superfície permite o desenvolvimento de projetos que extrapolam o que é tradicionalmente entendido por design de superfície. Um destes processos é o uso de design generativo, que permite o desenvolvimento de padrões e desenhos que seguem regras de construção complexas, e podem apresentar características não presentes tradicionalmente no design de superfície.

3. DESIGN GENERATIVO

Galanter (2003) nos traz o conceito da *generative art*, que nos auxilia a entender o que se entende por um sistema generativo, considerando o uso destes na criação de arte:

> *Generative art* se refere a qualquer prática de arte onde o artista utiliza um sistema generativo, como um conjunto de regras em linguagem natural, um programa de computador, uma máquina, ou outra invenção processual, que é posto em ação com um certa autonomia e contribui ou resulta em uma obra de arte concluída. (GALANTER, 2003, tradução nossa)

Extrapolamos desta definição então que, do mesmo modo que a *Generative Art* se refere à prática de arte com o uso de sistemas generativos, o *Generative Design,* ou Design Generativo, se caracteriza pelo design que faz uso das capacidades destes mesmos sistemas.

Seguindo a definição de Galanter (2003), Pearson (2011) simplifica o processo de criação generativo como: uma atividade que é uma maneira algorítmica de criar uma estética; uma colaboração entre um artista/designer e um sistema autônomo; um exercício em extrair resultados imprevisíveis de processos perfeitamente determinísticos; e uma busca por equilíbrio entre ordem e caos.

O filósofo Max Bense (2009) já concebia, em 1965, o que ele chamou de “estética gerativa”, que ele descreveu como uma “teoria matemático tecnológica da transformação de um *repertório* em *diretivas*, das *diretivas* em *procedimentos* e dos *procedimentos* em *realizações*” (BENSE, 2009, grifo do autor). Bense(2009) frisa ainda que neste ponto a obra não passa mais a ser imediata em relação ao criador, mas é mediada por um sistema de agregados semióticos e maquinais.

Um termo muito referenciado quando se fala em design generativo, é o termo *algoritmo* e cabe portanto encontrar uma definição. Cormen et al. (2001) definem algoritmo como “qualquer procedimento computacional bem definido que recebe um valor ou conjunto de valores como entrada e produz um valor ou conjunto de valores como saída. (Tradução nossa)”. Podemos então entender um algoritmo como um conjunto de regras que, quando executadas, transformam valores em resultados.

É importante porém salientar que o fato de um produto de design ser desenvolvido com base em um algoritmo ou um programa computacional não o classifica como um resultado de design generativo. Galanter (2003) destaca que a chave para se compreender a questão se dá em **como** o produto é feito, e não em por que ou quais são suas características depois de pronto. Galanter (2003) também afirma que para um sistema ser considerado generativo precisa ser “bem definido e auto contido o suficiente para operar com um grau de autonomia (Tradução nossa).”

Monro (2009) ainda aponta uma diferença sutil entre um sistema algorítmico e a

Arte — ou Design, considerando que o ímpeto definidor da atividade generativa está no processo generativo em si e não em seus motivos ou resultados — Generativa. De acordo com ele (2009) a atividade definida como algorítmica é a que produz com o uso de um algoritmo, que é executado (normalmente por um computador) e que quando finalizado resulta em um produto final. Este resultado não muda depois de feito. Já o processo chamado generativo inclui, além do uso de algoritmos como Galanter(2003) e Pearson(2011) já mencionavam, a possibilidade e a expectativa de manifestar elementos de “imprevisibilidade, comportamento em tempo real ou ambos (MONRO, 2009)”.

Segundo Bense (2009) o conjunto gerador, num sistema gerativo, transcorre seguindo o seguinte esquema:

Programa → (computador + gerador aleatório) → realizador.

O *programa* consistiria no conjunto de diretivas (indicações operacionais) expressas em um conjunto de signos da linguagem de programação usada. Ao *computador* cabe então executar os procedimentos algorítmicos que o *programa* determina e o *gerador aleatório* seria, então, denominado o princípio que permite também introduzir, nos procedimentos gerativos, fenômenos casuais, estes já previstos no *programa (BENSE, 2009)*.

É possível portanto entender por Design Generativo aquele design que é resultante — em todo ou em parte — de um processo generativo baseado em um sistema que possui um grau de autonomia, e que traz a possibilidade de comportamentos não previsíveis e/ou mutáveis, comportamentos estes que estão previstos dentro do algoritmo definido para este sistema.

3.1. SISTEMAS EMERGENTES E SISTEMAS COMPLEXOS

Uma questão de grande importância quando se trata de sistemas generativos é a da emergência, ou os “fenômenos casuais” de Bense(2009). Emergência é um termo amplo, que adquire diferentes significados em diferentes domínios do conhecimento, o tornando um termo de difícil definição. McCormack e Dorin (2001) indicam que a interpretação comum e não especializada do termo se refere a “ revelar, aparecer, ou tornar visível um evento, objeto ou resultado de um processo (tradução nossa)”. Eles (2001) também indicam que em um contexto de produção de arte — e design, neste caso — emergência pode também significar novidade e espontaneidade.

Malik (2014) associa sistemas emergentes à processos naturais, no ponto em que são resultantes de uma operação coletiva.

> Componentes individuais interagem para desenvolver efeitos complexos e de larga escala, que não são facilmente discerníveis a partir da operação individual destes componentes. Portanto, emergência descreve um sistema, o qual não pode ser previsto a partir das condições anteriores. Este já foi comparado com *resultantes*, que são resultados previsíveis, porém *emergentes* são aqueles efeitos ou resultados que não são previsíveis. (MALIK, 2014, tradução nossa, grifo nosso)

Os produtos de sistemas emergentes já causam interesse há séculos. Um exemplo disso são os *Musikalische Würfelspiele*, ou jogos de dados musicais, populares no século XVIII, eram uma forma de gerar músicas a partir de elementos pré-compostos ordenados de forma aleatória, através da rolagem de dados, por exemplo (NOGUCHI, 1997). Na área do design de superfície, podemos observar um exemplo similar em alguns trabalhos de Athos Bulcão, onde o designer é responsável pela criação do módulo gráfico, porém a organização final do padrão está livre de seu imediato controle, neste caso ficando a cargo do operário que o fixa na parede (Como o exemplo da figura 5).

Figura 5. Painel de azulejos no Instituto Rio Branco - Brasília, do Artista Athos Bulcão

Fonte: Edgard Cesar, 1998.

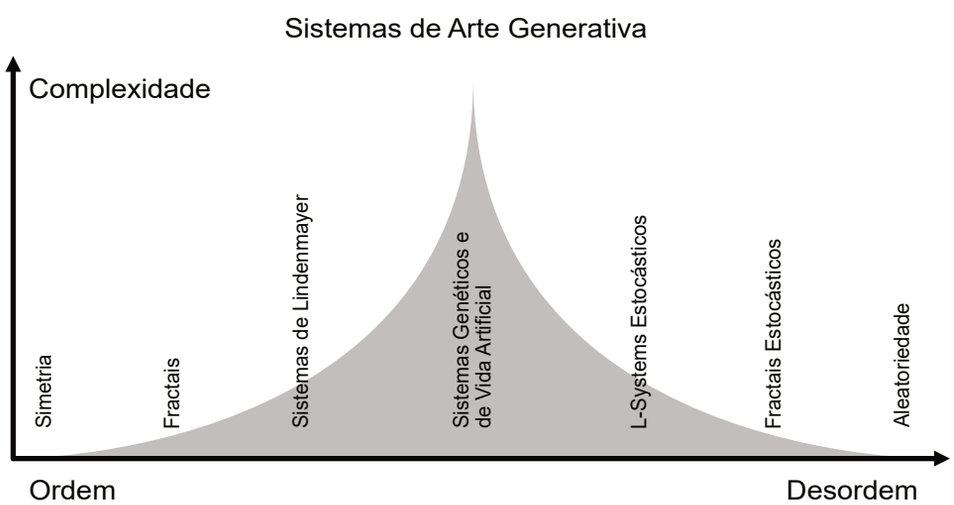
Em sua definição de sistemas emergentes, Malik (2014) vai ao encontro com Galanter (2003), que traça um paralelo entre a classificação de sistemas generativos e sistemas complexos dentro do que ele chama de “Teoria da complexidade(tradução nossa)”.

> Sistemas complexos tipicamente possuem um grande número de partes ou componentes menores que interagem com partes e componentes similares próximos. Estas interações locais frequentemente levam o sistema a organizar a si mesmo, sem qualquer controle máximo ou agente externo estar ‘no comando’. Tais sistemas são muitas vezes referidos como sendo auto-organizados. Estes sistemas auto-organizados são também sistemas dinâmicos sob constante mudança, e aquém de morte ou destruição, eles não se estabelecem em um estado final de ‘equilíbrio’. (GALANTER, 2003, tradução nossa)

Dada a associação que é possível perceber entre o uso de sistemas generativos para criação de imagens e os sistemas complexos, é possível compreender uma relação entre os sistemas generativos e certos sistemas naturais, que seguem princípios similares de emergência e complexidade. De modo similar ao que acontece com a biomimética, a inspiração em sistemas complexos naturais pode, e frequentemente traz uma fonte de inspiração para a definição de sistemas generativos.

Trazendo uma adaptação do trabalho de Gary Flakes no livro "*The Computational Beauty of Nature*", que analisa sistemas naturais através do viés computacional, Galanter(2003) nos apresenta uma relação entre o nível de complexidade de sistemas generativos e a quantidade de caos ou ordem associados a estes conforme a figura 6.

Figura 6. Complexidade e Sistemas generativos



Fonte: Traduzido pelo autor com base em Galanter, 2003.

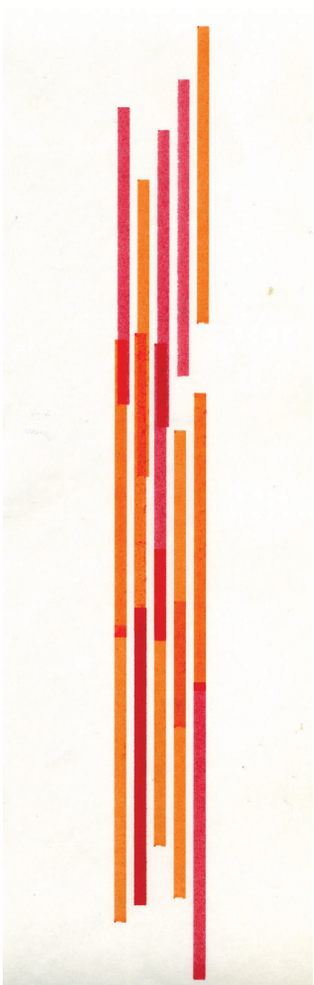
Os sistemas generativos complexos de fato são, então, aqueles que apresentam uma mistura efetiva de ordem e desordem, com sistemas generativos muito ordenados ou muito desordenados sendo considerados mais simples.

3.2. DESIGN GENERATIVO NA CONTEMPORANEIDADE

Apesar de não ser absolutamente necessário para a criação de sistemas generativos, a popularização dos computadores e a facilidade de acesso aos recursos de programação — para o aprendizado e publicação de código de software — tornou-os a principal ferramenta para criação de sistemas generativos. O interesse de explorar as capacidades de processamento dos computadores para fins estéticos iniciou já na década de 1960. O primeiro a publicamente apresentar sua Arte Algorítmica foi Georg Nees, com uma exposição na atual Universidade de Stuttgart

(GEORG, 2019). Outros nomes importantes no início da exploração da computação com propósitos estéticos são A. Michael Noll, Manfred Mohr, Roman Verostko e Vera Molnar, artista da peça apresentada na figura 8 (GEORG, 2019). Estes artistas utilizavam-se de diversas linguagens de programação diferentes, porém normalmente imprimiam suas composições por meio de plotters de mesa. Verostko chama o grupo de artistas que produzem utilizando algorítmos de Algoristas(VEROSTKO, 2019, tradução nossa).

Figura 7. Peça da obra Gothique, de Vera Molnar,desenho em plotter, tinta sobre papel, 1990.



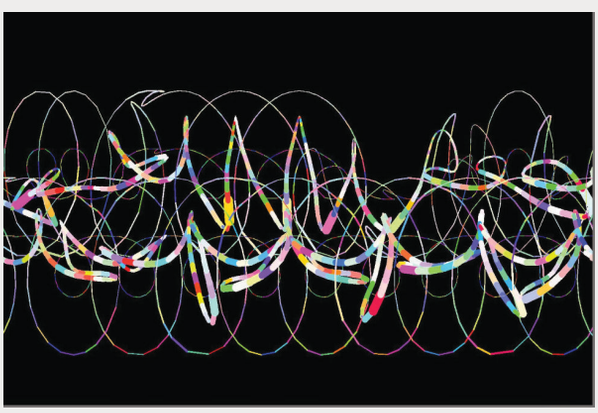
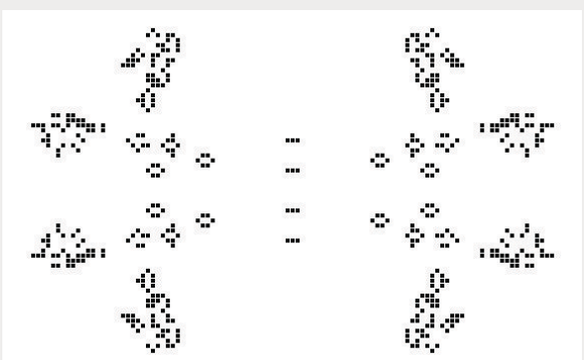
Fonte: Digital Art Museum, 2019.

Desde então a tecnologia e a capacidade de computação aumentou exponencialmente, permitindo novas explorações. A atividade se espalhou e é possível perceber o uso de sistemas generativos em diversas áreas. Na música a vertente eletrônica adotou técnicas generativas, tanto em um nível compositivo quanto em um nível menor onde os sistemas auxiliam na modulação do audio e alteração de timbres (GALANTER, 2003). A computação gráfica e a animação digital também possuem uma relação muito forte com as técnicas generativas.

Componentes que antes precisavam que o animador coreografasse todo o movimento, agora usam de comportamentos preestabelecidos para simular o mundo real. A popularização das linguagens de programação e das tecnologias de projeção também fizeram que a cultura *VJ*[1] adaptasse elementos generativos para uso em shows, clubes noturnos e outros cenários sociais (GALANTER, 2003).

No campo de criação generativa de imagens, Hobbs (2018) indica algumas das abordagens mais comuns usadas para gerar resultados emergentes são a simulação de *sistemas caóticos*, que geram caos e imprevisibilidade ao longo do tempo, como o pêndulo triplo, os *autômatos celulares*, que é um sistema de células organizadas em grade que podem estar ligadas ou desligadas dependendo do estado das células vizinhas. O exemplo mais notável de um algoritmo de autômatos celulares é o chamado Jogo da Vida, criado por John Norton Conway em 1970, que utiliza as propriedades de autômatos celulares de comparar células vizinhas para criar um sistema onde, a partir de um estado inicial, é possível observar uma evolução do estado conforme as regras definidas por Conway (GARDNER,1970). Os *sistemas de agentes*, ou atores, onde existe um conjunto de objetos com comportamentos preestabelecidos, e quando simulados interagem com o ambiente e entre si, e claro, os *geradores estocásticos* de valores.

Quadro 2. Abordagens de Sistemas Generativos de Imagem segundo Hobbs(2018)

	**´Triple Pendulum!** Mat Suarez, 2014 Abordagem de pêndulo triplo
	**Conway's Game of Life** Apps By Kids, 2018 Abordagem de Autômatos Celulares

1 . VJ: Vem de *Video Jockey,* e descreve o indivíduo que produz e manipula conteúdo visual em tempo real através de mediação técnica e tecnológica para uma audiência, frequentemente em ambientes de concertos musicais, clubes noturnos, festivais ou em associação com outras apresentações de arte performática. (WHAT IS VJING, 2019)

Process 18 (Software 3)

Casey Reas, 2010

Abordagem de Agentes Autônomos

Aquatics (detalhe)

Lieven Menschaert, 2019

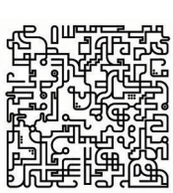
Abordagem de Geradores Estocásticos

Fonte: Elaborado pelo autor, 2019.

3.3. SUPERFÍCIES E DESIGN GENERATIVO

Como já mencionado a definição do que é considerado design de superfície generativo pode ser descrita como design de superfície criado com o uso de sistemas generativos. Um exemplo encontrado segue o princípio de criar módulos fixos e utilizar sistemas generativos para sua organização em padrões a ponto do padrão não se repetir em sua extensão. Um passo além, seria o modelo descrito por Russell (2014) onde a impressão — que neste caso precisa ser digital — produz um design que se altera enquanto está sendo impresso. Isto também poderia acarretar em um padrão não repetido em sua extensão, mas com possibilidades de alterar o design a ponto em que os próprios módulos se alterem. A criação de ilustrações baseadas em princípios generativos para uso em superfícies também pode ser considerada como um exemplo de aplicação bem sucedida na exploração da área.

Figura 8. Tiletoy, outro exemplo de padrão não-repetitivo, que pode ser gerado ao longo da impressão.



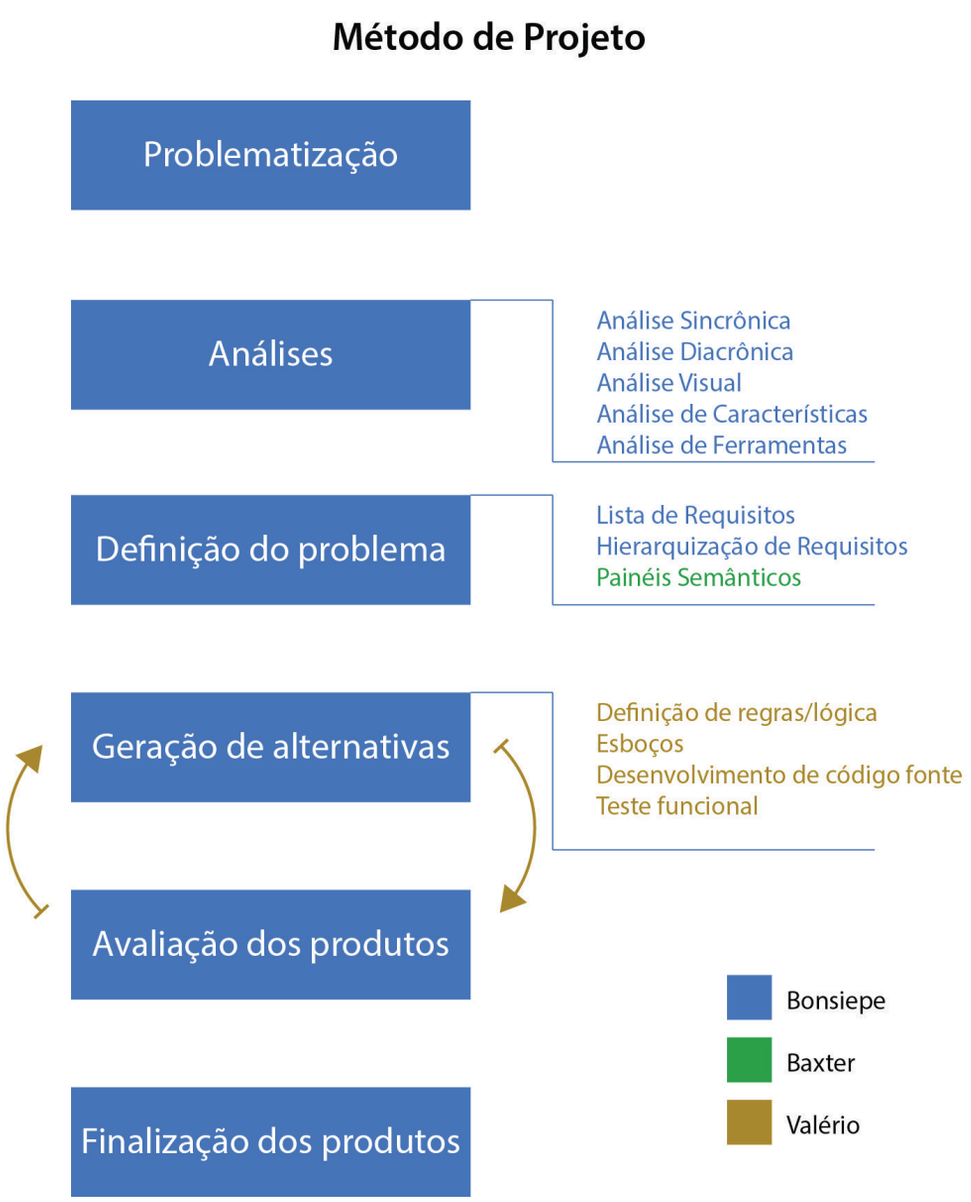
Fonte: MONOVEKTOR, 2011.

Sistemas generativos podem ser aplicados ao desenvolvimento de superfícies de diversas naturezas: superfícies estrutura (em associação ao design paramétrico), *projection mapping* sobre superfícies, superfícies têxteis, cerâmicas, papel e ainda superfícies digitais. Além do mais, o uso de sistemas generativos permite e influencia o designer de superfície a pensar as técnicas e ferramentas disponíveis de formas diferentes, permitindo um vocabulário maior ao projetar os seus produtos (LARANJEIRA; MARAR, 2019). O design generativo de superfície ainda inclui em suas características aspectos que acomodam a customização em massa, quando associado a tecnologias de produção compatíveis, como a manufatura aditiva ou métodos de impressão digitais, por exemplo, possibilitando uma valorização dos produtos obtidos do processo, pois cada usuário pode selecionar ou até influenciar o produto final, caso o processo generativo tiver sido desenvolvido com esta intenção.

4. DESENVOLVIMENTO DE SISTEMAS GENERATIVOS PARA SUPERFÍCIE

O método para a exploração das possibilidades de desenvolvimento adapta o modelo descrito por Bonsiepe(1984), com etapas adaptadas de Baxter (2017). O método também adapta parte do ciclo de desenvolvimento apresentado por Valério (2013) como “Processo de design gerativo”. Seguindo em ordem: Problematização e definição de objetivos, etapa de análises e estudos sobre os produtos a serem desenvolvidos e a temática a ser explorada, etapa de definição dos requisitos a serem atendidos, etapa de geração de alternativas e desenvolvimento em código, etapas de avaliação dos resultados — com possibilidade de retornar à etapa anterior para alterações — e etapa de finalização e formalização dos produtos.

Figura 9. Etapas de Desenvolvimento de Produtos Generativos



Fonte: Elaborado pelo autor com base em Bonsiepe (1984), Baxter (2017) e Valério (2013), 2018.

4.1. PROBLEMATIZAÇÃO

De acordo com os objetivos definidos, a problematização se define pela vontade de desenvolver quatro sistemas generativos de imagem, que possam ser utilizados para aplicação em superfícies, de forma estática tradicional na forma de tecido, papel e/ou cerâmica, ou ainda na forma animada, mutável em tempo real na forma de superfícies virtuais ou em projeções digitais sobre superfície. Fica definido também que, para explorar de forma satisfatória as possibilidades de um sistema generativo, se buscou inspiração em comportamentos complexos existentes na natureza, nesse caso, comportamentos de grupo de diferentes animais que produzem efeitos passíveis de explorar para fins estéticos.

Quadro 3. Problematização

O QUÊ?	POR QUÊ?	COMO?
Conjunto de Sistemas Generativos de Imagem para uso em design de superfície	Possibilidades ainda pouco exploradas, tanto como resultados quanto processos projetuais	Programação digital de Sistemas Generativos

Fonte: Elaborado pelo autor, 2019.

4.2. ANÁLISES

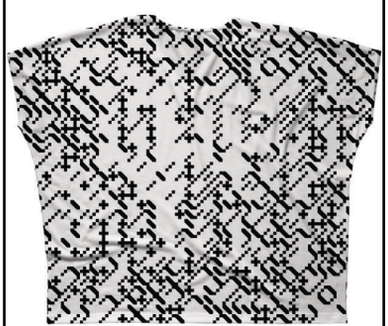
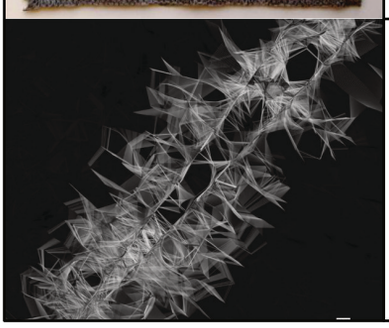
Na *análise sincrônica* foram comparados diferentes sistemas generativos e suas características. Na *análise diacrônica* foram estudados diferentes sistemas generativos através do tempo. Na *análise de ambientes de desenvolvimento* foram elencados alguns dos principais ambientes de desenvolvimento e programação utilizados na criação de sistemas generativos digitais de imagem. E finalmente, na *análise morfológica* ou para este projeto, *análise visual dos comportamentos*, foram estudadas características de diferentes comportamentos de grupo naturais com fim de compreender como um sistema inspirado em tais deve se comportar visualmente.

4.2.1. Análise Sincrônica

Seguindo a metodologia de projeto adotada, foi executada a análise de um conjunto de sistemas de design generativo existentes no mercado para verificar como estes se comportam em um ambiente de mercado, quais são suas características e como estas implicam em seu sucesso como sistema generativo e como produto. Segundo Bonsiepe (1984), a análise sincrônica seve para “reconhecer o ‘universo’ do produto em questão e evitar reinvenções. A comparação e crítica dos produtos requer a formulação de critérios comuns.” Foram elencadas 3 características de cada

sistema analisado. São estas: plataforma, aplicação/propósito final e abordagem utilizada.

Quadro 4. Levantamento de alguns sistemas generativos existentes.

	Blusa Contrast Set Fonte: Contrast.parts, 2018. •Desenvolvido em Processing •Estampa corrida sem repetição •Gerador estocástico para seleção de módulos
	Orbitals a (detalhe) Fonte: Complexification, 2004. •Desenvolvido em Processing •Painel Digital •Diversos sistemas similares a pêndulos, se movendo ao longo do tempo
	Cellular Automata Fonte: Turner, 2012. •Tricô manual •Painel •Regras inspiradas na lógica de Autônomos Celulares
	Process 19 (software 2) Fonte: Reas, 2014. •Desenvolvido em Processing •Superfície animada bidimensional •Sistema de Agentes, onde colisões/relações entre objetos resulta em efeitos visuais

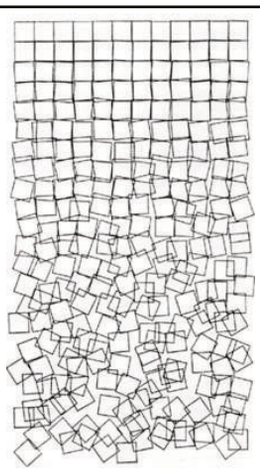
Fonte: Elaborado pelo autor, 2019.

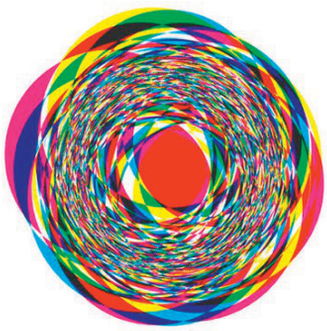
É possível notar certa simplicidade em termos de módulos e paletas de cores nos exemplos analisados. É possível extrapolar que esta simplicidade serve para facilitar ou otimizar recursos no momento de geração de cada design, como tempo para o trabalho manual ou capacidade de processamento nos trabalhos de geração digital.

4.2.2. Análise Diacrônica

Na análise diacrônica, foram escolhidos trabalhos generativos criados a partir de programação de computador, para limitar a amplitude da análise e obter mais facilidade de associar e comparar as características percebidas com o presente projeto. Segundo Bonsiepe (1984) a análise diacrônica seria “uma coleção de material histórico para demonstrar as mutações do produto no transcurso do tempo.” Foram escolhidos trabalhos que podem não ter sido classificados como generativos quando criados/expostos, mas que ainda assim seguem a classificação encontrada na etapa de fundamentação teórica desta pesquisa. São elencados para estes trabalhos: ano, tecnologia utilizada, autoria e aplicação/finalidade.

Quadro 5. Levantamento de sistemas generativos de imagem ao longo dos anos.

	Schotter (1968) de Georg Nees Fonte: compart, 2019. •Desenvolvido em Algol e impresso em plotter de mesa •Obra de arte
	Structures de quadrilatères (1986) de Vera Molnar Fonte: Digital Art Museum, 2019. •Desenvolvido em Fortran •Obra de arte

	Fireball (2006) de John Maeda Fonte: symplyjohnmaeda, 2019, •N/A •Ilustração para capa de livro
	Fragments of vision (2018) de Tyler Hobbs Fonte: Hobbs, 2019. •Desenvolvido em Clojure e impresso com jato de tinta •Obra de arte

Fonte: Elaborado pelo autor, 2019.

Existe uma clara evolução entre os exemplos analisados no que diz respeito a quão intrincada a composição, assim como o número de objetos e cores processados ao mesmo tempo. Parte dessa evolução se dá claramente pelas limitações tecnológicas das obras mais antigas em comparação com os exemplos mais recentes, porém os exemplos mais novos também tem a vantagem do conhecimento estabelecido e da exploração que aconteceu no campo entre o início do uso de computadores para criação de imagens generativas e as obras mais recentes.

4.2.3. Análise de características dos sistemas generativos

Nesta etapa foi feita uma lista de características frequentemente associadas a esse tipo de produto. Foram listadas características consideradas pertinentes a este projeto, que foram reveladas através da pesquisa bibliográfica executada.

Quadro 6. Lista de Características

Lista de Características - Sistema Generativo
• Sistema de regras, executadas com um nível de autonomia
• Resultados obtidos não são imediatos ao designer, mas intermediados
• Complexidade advinda do uso de elementos caóticos associados a elementos de ordem

• Resultados emergentes e imprevisíveis
• Possíveis resultados complexos descendem da operação conjunta e interação entre elementos menores dentro do sistema.
• Tendência à simulação de efeitos e comportamentos percebidos na natureza
• Pode ser utilizado para criação de imagens estáticas, animações ou ainda produtos interativos, na área do design de superfície.
• Geralmente programado utilizando uma linguagem de programação digital, que é então interpretada pelo computador para execução das regras definidas

Fonte: Desenvolvido pelo autor, 2019.

Através desta lista fica facilitado o elencamento de requisitos para o projeto, pois fica mais clara o que se atribui ao produto como característica assim como o que não é pertinente considerar como necessário para o sucesso do projeto.

4.2.4. Análise de Ambientes de Desenvolvimento

Existem disponíveis diversas ferramentas para auxiliar a programação de um sistema generativo de imagens, portanto viu-se necessário analisar quais as vantagens e desvantagens de cada ambiente de desenvolvimento, assim como possíveis custos. Desta análise define-se então qual o ambiente de desenvolvimento será utilizado para criar os produtos do projeto.

Quadro 7. Análise dos ambientes de desenvolvimento

Plataforma	**Preço**	**Compatibilidade**	**Disponibilidade de recursos (tutoriais)**	**Outras Características**
Processing	Gratuito/ Open-Source	PC, Mac, iOS, Android, HTML5, Linux	Expansiva	Ambiente flexível e bem estabelecido
Context Free Art	Gratuito	PC, Mac, Linux	Baixa	Ambiente simples, porém limitado
vvvv	Gratuito/ Licença CC	PC	Baixa	Programação em nós completa, não-comercial
openFrameworks	Gratuito/ Open-Source	PC, Mac, iOS, Android, HTML5, Linux	Média	Baseado em C++ (linguagem complexa)
Cinder	Gratuito/ Open-Source	PC, Mac, Linux	Média/Baixa	Baseado em C++ (linguagem complexa)
NodeBox	Gratuito/ Open-Source	PC, Mac, Linux	Média	Ambiente simples, mas completo Programação em nós

Fonte: Elaborado pelo autor, 2019.

Considerando a necessidade de aprendizado e adequação ao ambiente de desenvolvimento durante o projeto, o fato de plataforma Processing ter uma expansiva coleção de recursos para aclimatação ao ambiente de desenvolvimento o torna a opção mais viável. Outras características que fizeram o ambiente Processing ser escolhido para o projeto são sua flexibilidade e possibilidade de programação modular. Processing é uma linguagem de programação que nasceu do programa de Estética e Computação do laboratório de mídia do Instituto de Tecnologia de Massachussets, e foi criada por Ben Fry e Casey Reas para simplificar o processo de programação para criadores visuais.(Greenberg, 2007)

4.2.5. Análise visual dos comportamentos naturais de bando

Nesta análise foram escolhidos alguns exemplos de comportamentos na natureza para serem avaliados em relação às suas características visuais, para inspirar e auxiliar na criação dos produtos. Similar ao que Bonsiepe (1984) descreve como Análise Morfológica, esta etapa “serve para reconhecer a estrutura formal de um produto, [...] sua composição,” embora neste caso o que está a ser analisado são os comportamentos que irão inspirar as simulações generativas posteriores e suas estruturas formais. Os comportamentos analisados foram: um cardume se agrupando contra predadores, um grupo de borboletas migrando, um rebanho de ovelhas entrando em um cercado e um murmúrio de estorninhos.

4.2.5.1. Peixes

O cardume em formação defensiva se mantêm em um único grupo, com alta proximidade entre os indivíduos, e realizando um movimento circular conjunto. A velocidade dos indivíduos é uniforme, e estes não se separam da massa formada pelo grupo. A forma tridimensional do grupo é amorfa, com ondulações criadas por pequenas variações na velocidade dos indivíduos. A luz reflete diretamente em cerca de metade dos indivíduos do grupo (figura 10).

Figura 10. Cardume de anchovas em formação defensiva.

Fonte: BBC Earth, 2017.

4.2.5.2. Borboletas

O comportamento das borboletas, por ser migratório, apresenta uma direção geral. Não existe uma distância mínima respeitada entre os indivíduos, e o caminho seguido parece ser orientado parcialmente por correntes de ar, gerando uma impressão de fluxo de movimento. O fato de ser um grande volume de indivíduos voando na mesma direção, forma a impressão de camadas, por causa dos efeitos de paralaxe existente (figura 11).

Figura 11. Borboletas monarca em migração.

Fonte: The HDR Channel, 2017.

4.2.5.3. Ovelhas

O rebanho apresenta velocidade individual uniforme e coesão, porém pouco alinhamento, no sentido em que existe uma certa facilidade de novas direções de movimento serem adotadas por indivíduos. O alinhamento porém aumenta quando o grupo está se deslocando em direção à porteira assim como a separação entre os indivíduos diminui quando estes estão cruzando a porteira. O agrupamento entre os indivíduos é muito próximo, e esporadicamente indivíduos se separam do grupo principal (figura 12).

Figura 12. Rebanho de ovelhas adentrando cercado.

Fonte: Caters Clips, 2016.

4.2.5.4. Estorninhos

No murmúrio de estorninhos os indivíduos não se encostam, e a distância entre eles se altera conforme sua velocidade. Apesar de se manterem em uma massa única, existem diferenças notáveis entre partes desta mesma massa, seja em separação dos indivíduos ou velocidade. Quando existe uma aceleração na velocidade dos indivíduos do grupo, ela tende a acontecer de forma brusca. A impressão que se têm é de que o grupo segue um fluxo circular, porém com alterações inesperadas, que geram ondas de movimento dentro da própria massa de indivíduos. Por serem pequenos e o grupo ser grande, existem momentos em que os indivíduos parecem desaparecer ou diminuir. Esta ilusão se dá pela alteração no perfil dos indivíduos em decorrência de alterações na direção do voo. A diferença de tamanho percebida entre os indivíduos do grupo é sutil, e pode ser percebida principalmente quando uma parte da massa amorfa do grupo se sobrepõe sobre outra parte do mesmo grupo.

Figura 13. Murmúrio de estorninhos.

Fonte: Dylan Winter, 2010.

4.3. REQUISITOS E HIERARQUIZAÇÃO DE REQUISITOS

Segundo Bonsiepe(1984), os requisitos servem para orientar o processo projetual em relação às metas a serem perseguidas, assim como problemas e incompatibilidades a serem evitadas. Em seguida, a hierarquização destes requisitos serve para estabelecer prioridades no atendimento dos requisitos definidos, considerando a possibilidade de um requisito se apresentar incompatível com outro.

Quadro 8. Hierarquização dos Requisitos

Como DEVE ser	**Como SE ESPERA que seja**
• Inspirado em Comportamentos Naturais • Possibilidade de extração de imagens para impressão	• Desenvolvido em Processing • Confortável visualmente • Simplicidade nos módulos/agentes, complexidade nas interações e comportamentos destes • Criado com a abordagem de agentes/atores
Como NÃO PODE ser	
• Previsibilidade de comportamentos • Recorrência de estados visuais	

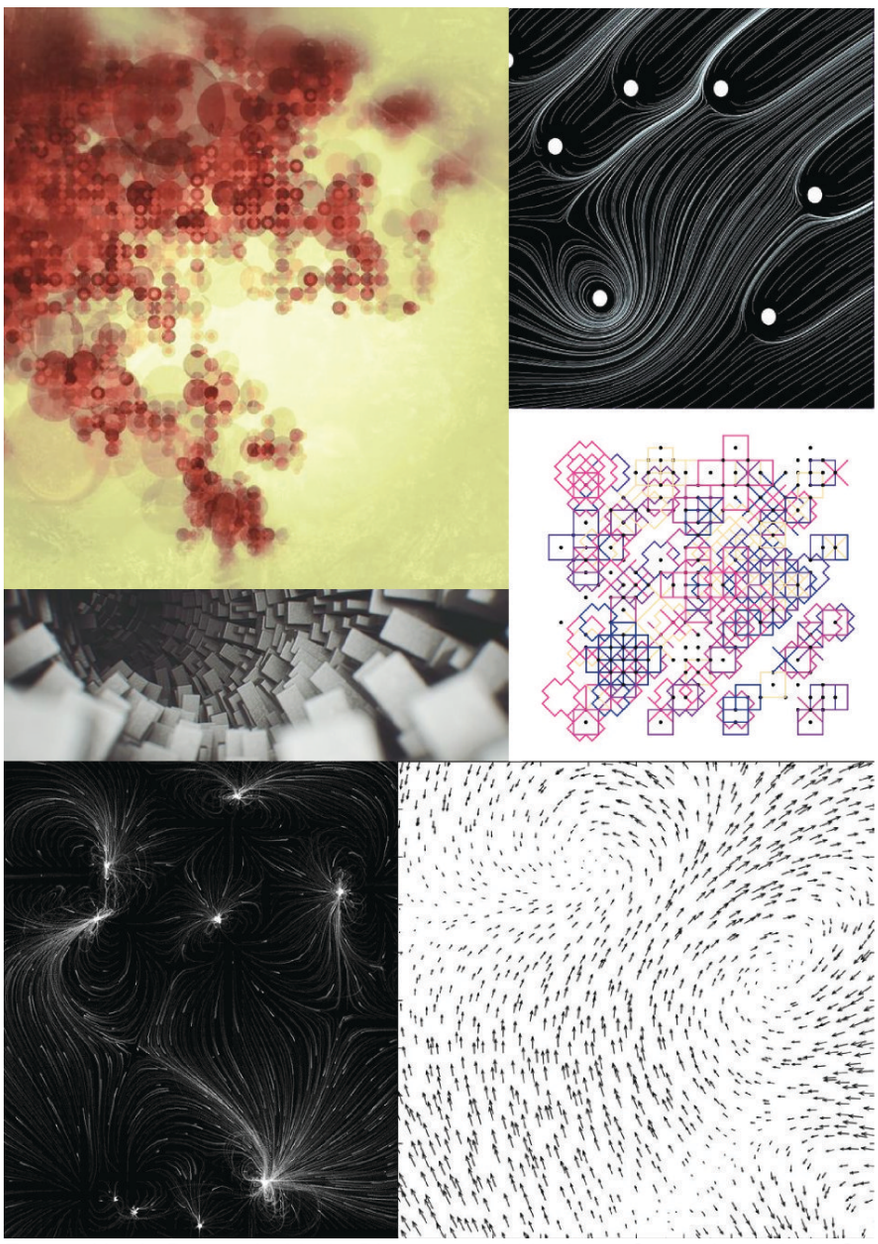
Fonte: Elaborado pelo autor, 2019.

4.4. PAINÉIS SEMÂNTICOS

A fim de direcionar o estilo visual a ser seguido pelos sistemas generativos de imagem e visualizar alguns exemplos e conceitos pertinentes ao projeto, foram desenvolvidos painéis semânticos, inspirados por Baxter (2017). A técnica de Baxter (2017) indica o uso de um painel de Estilo de Vida, que apresenta o público alvo, um painel de Sensação, que apresenta uma expressão de valor a ser associada ao produto a ser desenvolvido e, por fim, um painel de Tema Visual, que apresenta produtos que atingiram a sensação apresentada no painel anterior. Como adaptação à técnica de Baxter (2017), o painel Estilo de Vida foi ignorado, considerando o fato que o projeto não conta com um público alvo definido. Além disso, os painéis de Sensação e Tema visual foram adaptados para opções paralelas, mais relevantes ao projeto. Os painéis que foram desenvolvidos descrevem *Estilo e Linguagem Visual* e *Comportamento e Movimento,* como paralelos aos painéis Tema Visual e Sensação de Baxter(2017), respectivamente.

O painel de **Estilo e Linguagem Visual** (figura 14) identifica algumas imagens criadas por sistemas generativos, possibilitando visualizar algumas das possibilidades formais e estilísticas dentro do meio do design generativo digital.

Figura 14. Painel Estilo e Linguagem Visual



Fonte: Noah Paessel, 2013, Pinterest, 2019, Joey Camacho, 2014, Saskia Freeke, 2019, Corneel Cannaerts, 2011, Laidlaw et al., 2001 .

No painel de Estilo e Linguagem Visual é possível perceber uma certa economia de cores, assim como simplicidade nos módulos. Essa tendência pode ser entendida como um modo de aliviar ou facilitar custos em processamento computacional, permitindo que mais módulos sejam simulados ao mesmo tempo.

O painel **Comportamento e Movimento** (figura 15) apresenta exemplos de comportamentos naturais em estados estéticos bem-sucedidos, auxiliando na composição final dos elementos do produto. O painel também apresenta elementos externos ao comportamento imediato dos animais, mas que, fazendo parte do ambiente onde estes estão inseridos, alteram e complementam este.

Figura 15. Painel Comportamento e Movimento

Fonte: The HDR Channel, 2017, BBC Earth, 2017, Caters Clips, 2016, Dylan Winter, 2010.

No painel de Comportamento e Movimento é possível perceber que as formas que os grupos tomam são orgânicas e amorfas, porém todas possuem uma direcionalidade.

4.5. GERAÇÃO DE ALTERNATIVAS E DESENVOLVIMENTO DOS PROGRAMAS

Pela natureza do trabalho a ser feito e a necessidade de simular diversos indivíduos simultaneamente, a abordagem a ser seguida para a programação dos sistemas generativos é a de agentes, ou atores como apresentados por Hobbs (2018).

Dentro desta abordagem, um sistema frequentemente usado adota as ideias de Reynolds (1999), de comportamentos de direção para agentes autônomos. Reynolds (1999) apresenta uma solução para agentes digitais que apresentam a capacidade de navegar seu ambiente de forma realista e improvisada. Agentes neste caso descreve um conjunto de módulos digitais, aos quais se aplicam funções de movimento, aparência e comportamento. Para este efeito, Reynolds (1999) define que cada agente deve possuir uma capacidade limitada de perceber seu ambiente, processar esta informação e calcular uma ação a ser tomada.

Neste projeto esta ação se caracteriza por uma força de movimento aplicada ao próprio agente em forma de cálculo vetorial, que Reynolds (1999) chama de “Velocidade Desejada”. O movimento de cada agente dentro de um sistema é então definido pela soma de objetivos atribuídos a cada agente, com diferentes prioridades e que aplicam diferentes forças direcionais e que resultam em uma “velocidade desejada” única, que é o resultado da soma de todos os vetores vindos de cada objetivo. Esta “velocidade desejada” final é então aplicada ao motor de movimento usado na programação para influenciar a direção de cada agente, individualmente. Este motor calcula a velocidade de cada indivíduo a cada quadro de animação utilizando três principais vetores: **posição, velocidade e aceleração.** O vetor posição descreve a posição do indivíduo no plano cartesiano de pixels usado no programa. O vetor velocidade descreve quantos pixels por quadro de animação o indivíduo se desloca, assim como a direção deste deslocamento. E por fim o vetor aceleração descreve uma força aplicada ao indivíduo, que vai resultar numa alteração da velocidade. No programa esta operação funciona por adição destes vetores a cada quadro de animação, onde o vetor de aceleração é adicionado ao vetor de velocidade, que por sua vez é adicionado ao vetor de posição. É comum nesta operação limitar a velocidade após a adição do vetor de aceleração para evitar comportamentos imprevisíveis ou velocidades que não condizem com o comportamento que se deseja criar. Após a adição da velocidade ao vetor de posição a aceleração é reduzida a zero, para não influenciar os cálculos de velocidade posteriores, que podem não estar sofrendo influências de forças aceleradoras a todo instante.

O trabalho criativo se caracteriza então, por definir quais são os “objetivos” do grupo de agentes e implementar no código os comportamentos que surgem a partir destes. Alguns comportamentos já foram descritos por Reynolds (1999, tradução nossa) em seu artigo, como **Buscar**, que guia o agente a um objetivo da forma mais direta possível, **Alinhar,** que dirige o agente para a direção média dos agentes ao seu redor, **Coesão,** que direciona o agente para a média da posição dos agentes ao seu redor formando um grupo unificado, **Separação,** que faz com que os agentes não ocupem o mesmo ponto no espaço, entre outros comportamentos. Além dos

comportamentos descritos por Reynolds (1999), existe a possibilidade de criar outros comportamentos, utilizando o princípio de cálculo de “velocidade desejada”.

4.5.1. Definição de Lógica Básica

Para traduzir os esboços em programas de computador, foi criado um código base, que segue a lógica de comportamentos de Reynolds (1999), que é então alterado e ampliado para cada simulação, conforme apresentam-se as necessidades.

A linguagem Processing é uma linguagem de de programação orientada a objetos, o que significa que ela possui um mecanismo de definições de *Class*, ou classe, que funciona como um diagrama de um módulo de programação. Neste diagrama é possível definir as características associadas àquele módulo, sejam estas características visuais, funcionais ou informacionais. No programa, então, é possível criar instâncias (objetos criados com base no diagrama definido pela classe) deste módulo e utilizar suas próprias características funcionais ou ainda chamar funções externas e aplicá-las a cada instância do módulo em questão. Processing executa seus programas dinâmicos (aqueles onde existe animações e os resultados do programa não são estáticos) com base em duas funções básicas, **setup** e **draw.** Setup contém as configurações que são interpretadas no início da execução do programa, tais como tamanho da janela onde o programa é executado, quantidade de quadros por segundo e qualquer outra função que deva ser executada apenas uma vez na inicialização do programa. Draw já é a função que acomoda todas as funcionalidades e atribuições que precisam acontecer recorrentemente. É nesta função onde são aplicadas as animações e os comportamentos dos objetos. Tudo dentro da função draw acontece uma vez para cada quadro de animação.

Utilizando Processing e a possibilidade de programação orientada a objetos, que possibilita a reutilização de blocos de código, assim como aplicação modular destes, foi criada uma *Class* chamada Animal, que abriga todas as varáveis e funções básicas necessárias para a simulação dos comportamentos definidos.

As principais funções básicas escritas na classe são:

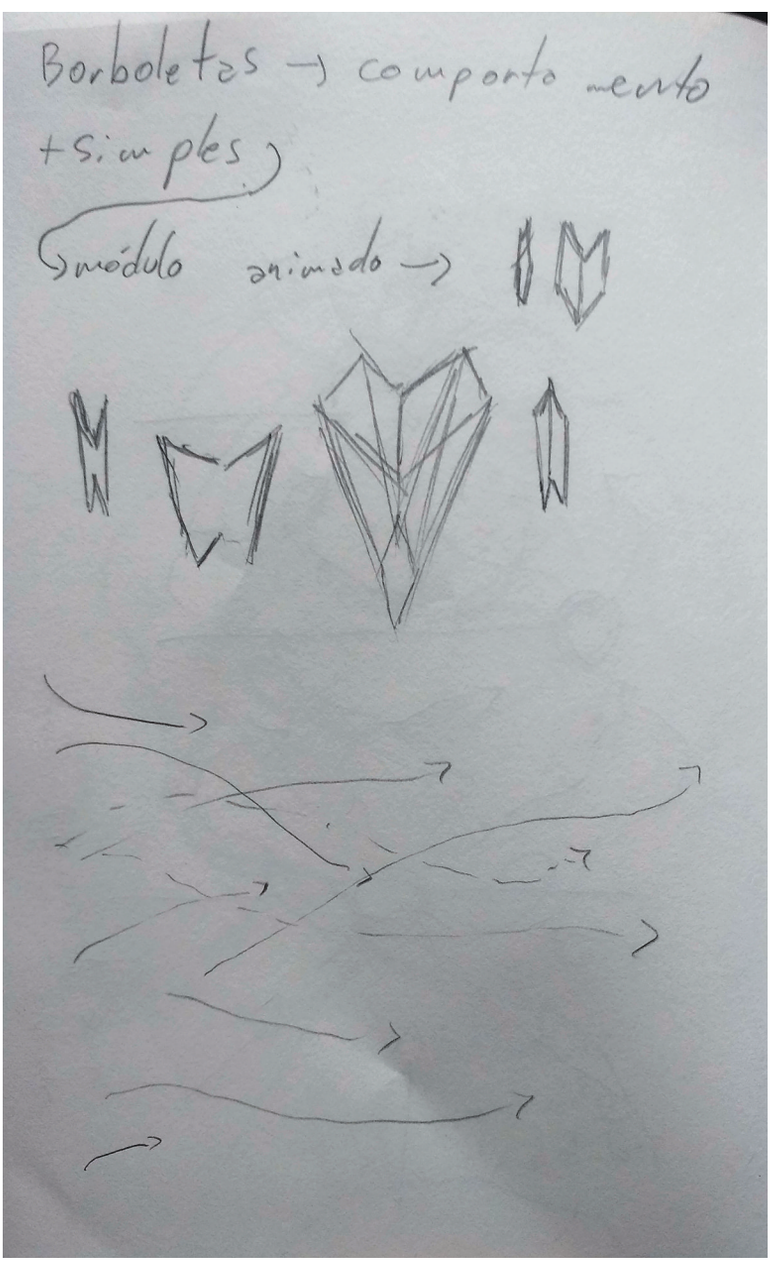
- **flock:** Define os comportamentos de separação dos outros indivíduos, alinhamento de movimento e coesão na localização
- **applyForce:** Facilita a aplicação de forças motrizes ao movimento de cada indivíduo.
- **repel**: Repele obstáculos, fazendo com que os indivíduos fujam deste.
- **seek**: Permite a definição de um objetivo, ou “ponto final” para ser buscado pelos indivíduos.
- **render:** Descreve e aplica a aparência visual das unidades.
- **followField:** Força cada indivíduo a seguir um campo de fluxo predefinido.

Após a definição da lógica e do código base, inicia-se o desenvolvimento dos programas específicos. Parte das figuras que ilustram as etapas do processo de desenvolvimento de cada programa possuem um *QRcode* que leva a um breve vídeo que ilustra o comportamento animado para facilitar a compreensão das etapas.

4.5.2. Borboletas em migração

Para iniciar o desenvolvimento do programa que simula a migração de borboletas, foram criados alguns esboços de módulos para criar no código. O fluxo e o movimento deste programa é comparativamente simples, pois o movimento geral segue uma única direção, com apenas pequenas variações.

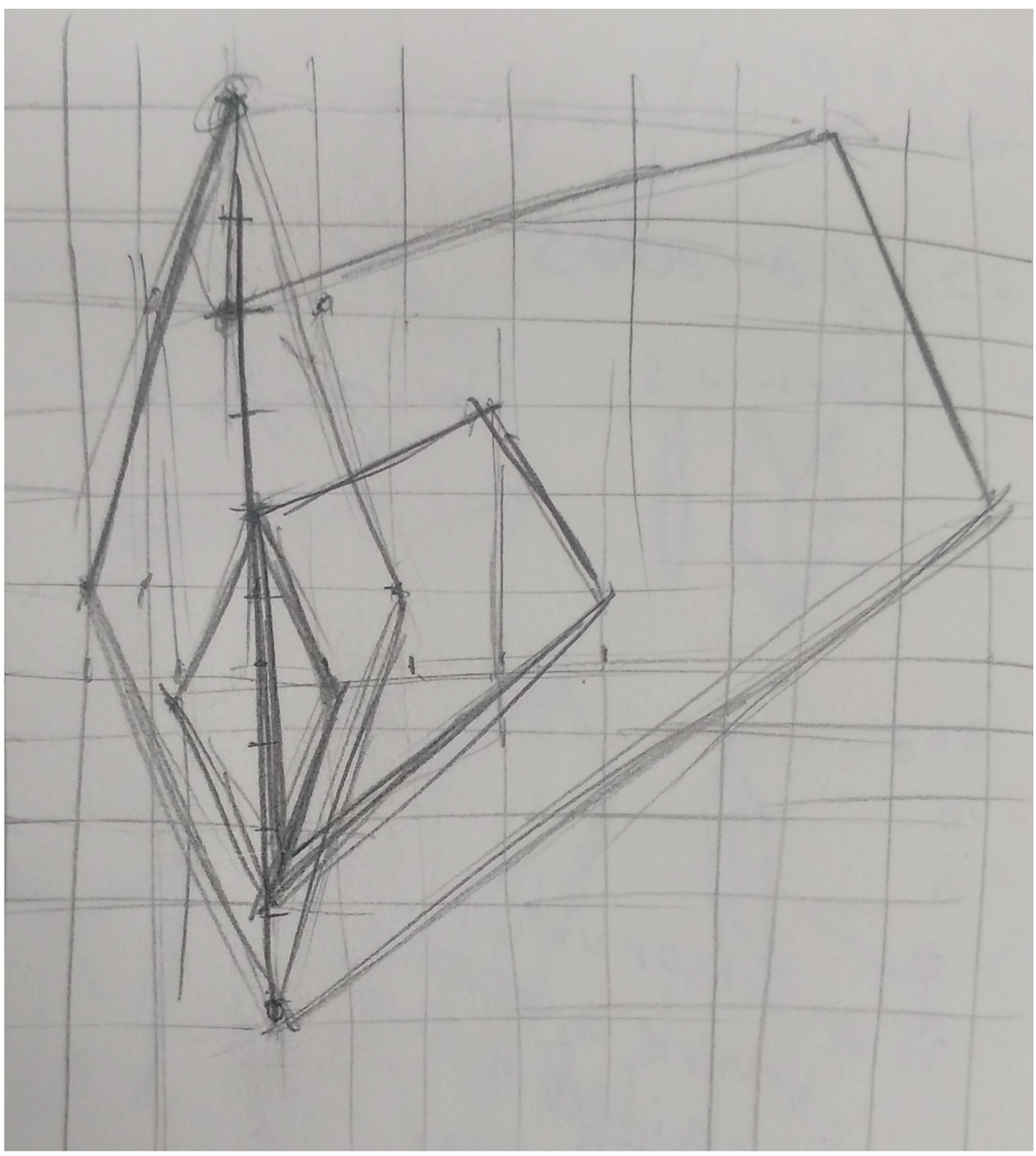
Figura 16. Simulação Borboletas.



Fonte: Elaborado pelo autor, 2019.

Além do esboço, foi criado um diagrama para facilitar o desenho por código do módulo, assim como guiar a definição da animação do módulo, que se dá pela interpolação dos vértices do desenho geométrico do módulo.

Figura 17. Diagrama Borboleta.



Fonte: Elaborado pelo autor, 2019.

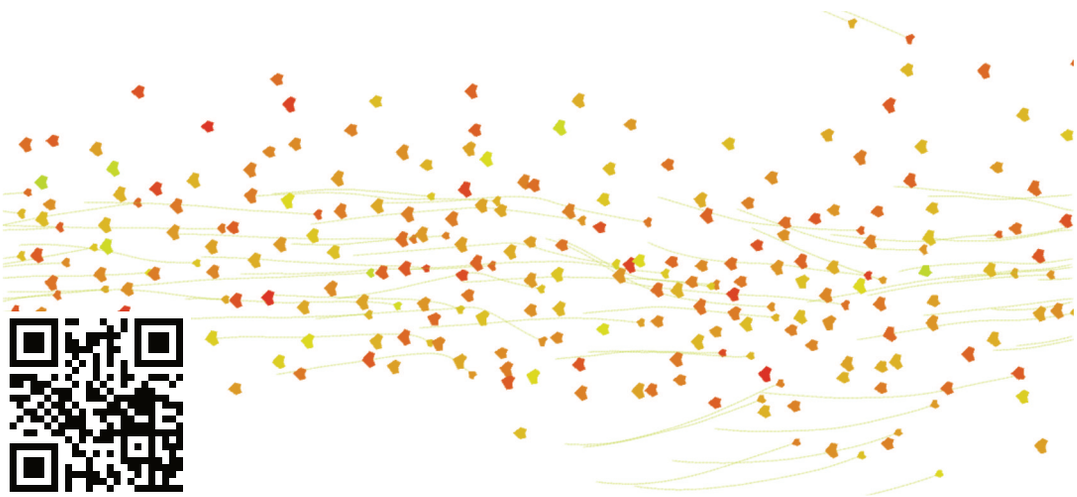
A partir do diagrama, foi escrito o código que desenha o módulo a partir dos vértices da forma geométrica, onde cada ponto no plano cartesiano corresponde a um vértice localizado no plano de pixels do programa. Foi escrita também a função que anima o módulo através da interpolação dos vértices da figura, para simular o bater de asas da borboleta.

Figura 18. Desenho por vértices.

Fonte: Elaborado pelo autor,2019.

Depois de definida a aparência do módulo, o código do desenho e da animação foi implementado ao código base, e este foi configurado para obter um comportamento semelhante ao comportamento observado, com alta separação entre indivíduos, alto alinhamento, baixa coesão e direção uniforme. Além do comportamento base, foi criado um comportamento de **evasão**, que faz com que os indivíduos desviem de objetos móveis invisíveis predefinidos, para simular correntes de vento no caminho do grupo. Foram também criadas duas camadas de indivíduos para simular o efeito de paralaxe, e dar a impressão de profundidade e volume. Nesta etapa também foram exploradas possibilidades de cores aleatórias e outros detalhes visuais para reforçar a ideia de fluxo visual, como rastros de movimento para alguns dos indivíduos (figura 19).

Figura 19. Protótipo de comportamentos.



Fonte: Elaborado pelo autor, 2019.

Concluída a definição do comportamento dos módulos, a atenção se volta para definições de natureza estética e visual, como a criação de uma paleta de cores(figura 20) baseada no painel semântico Comportamento e Movimento, o ajuste da velocidade da animação do bater de asas das borboletas e do formato final do rastro de movimento.

Figura 20. Paleta de cores para programa Borboletas.

Cores dos Módulos Cor de fundo

Fonte: Elaborado pelo autor, 2019.

Após a criação da paleta de cores, foi criada uma operação condicional para atribuir uma das quatro cores a cada módulo de forma aleatória. Esta operação foi aplicada também para atribuir cores aos rastros de movimento. Foi também aplicada uma cor de fundo para o programa (figura 21).

Figura 21. Programa Borboletas em migração.

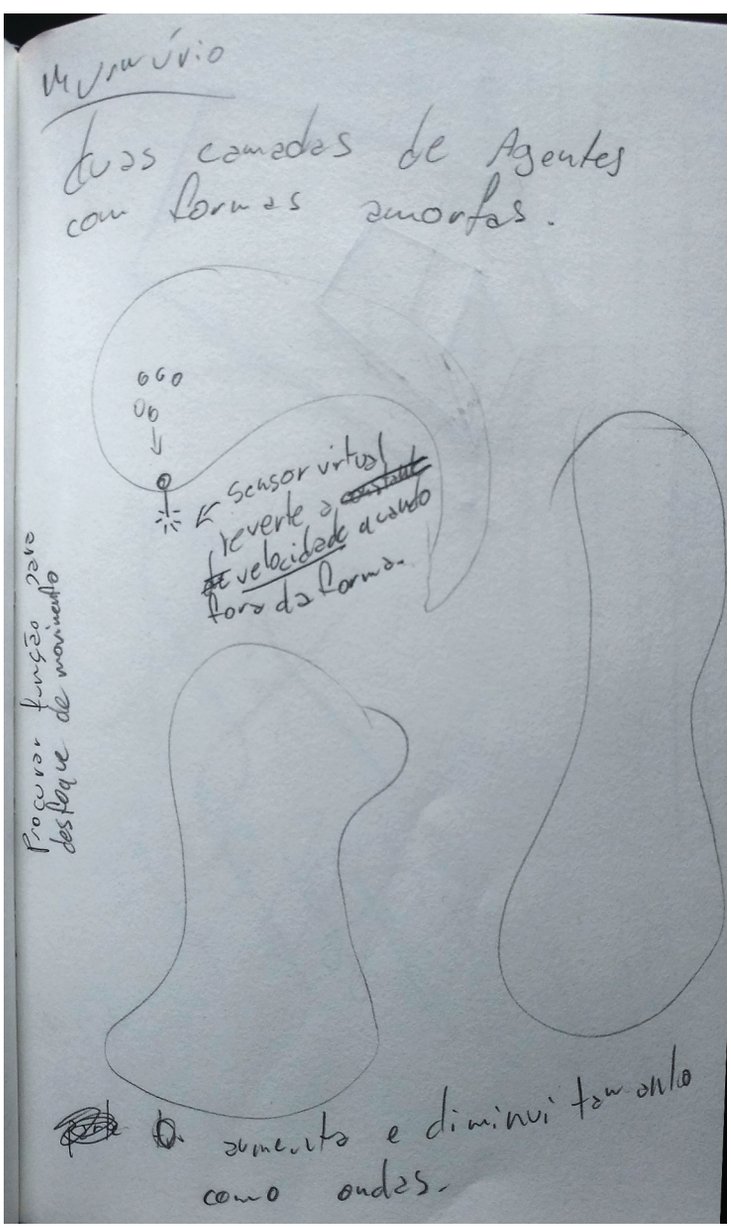
Fonte: Elaborado pelo autor, 2019.

A partir deste ponto o programa é capaz de gerar padrões baseados no comportamento definido e estes podem ser aplicados como projeções digitais animadas sobre superfície ou podem ser extraídos para serem impressos por meios tradicionais como um rapport.

4.5.3. Murmúrio de Estorninhos

O esboço para este programa foca em definir a forma da massa do grupo, e como as camadas se sobrepõem. A primeira ideia segue o uso de ondas para definir o formato de cada camada. Nesta ideia existe a possibilidade de usar sensores virtuais para detectar a posição de cada módulo, e manter estes dentro da forma definida (figura 22).

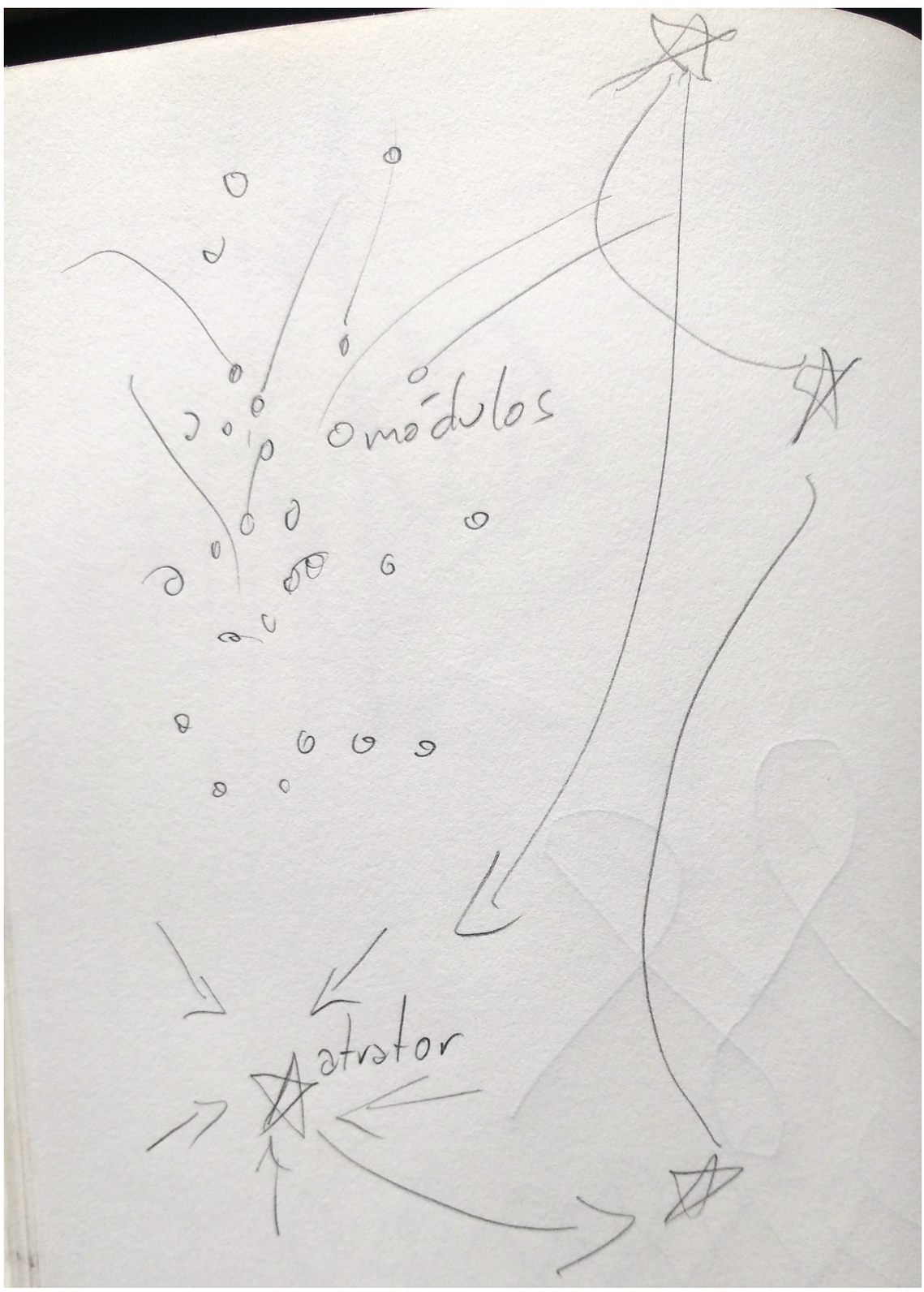
Figura 22. Simulação Murmúrio 1



Fonte: Elaborado pelo autor, 2019.

No esboço da figura 23 o movimento de cada camada de indivíduos segue um ponto específico que se movimenta no espaço, de forma a manter o comportamento semelhante ao observado.

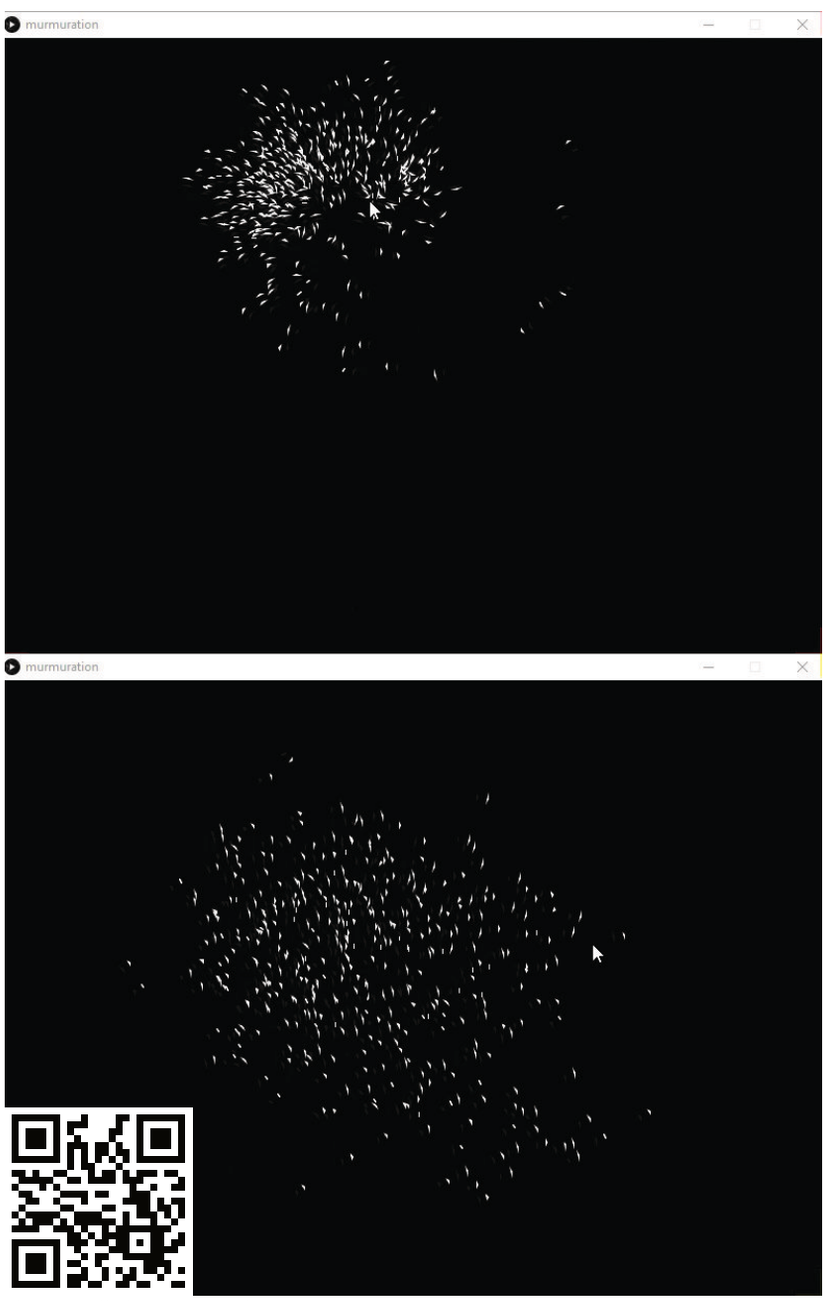
Figura 23. Simulação Murmúrio 2.



Fonte: Elaborado pelo autor, 2019.

Novamente foi criado um programa com a lógica base inspirada por Reynolds(1999). Neste caso foi usada a segunda ideia esboçada como base para a lógica, onde o grupo se desloca seguindo um ponto atrator que se move ao redor da tela. Para testar o funcionamento do programa com base no comportamento de busca de atrator foi criado um protótipo utilizando a posição do mouse como ponto de convergência, como na figura 24. O protótipo foi criado com dois grupos diferentes de indivíduos que se direcionam ao atrator, porém um grupo não influencia o comportamento do outro, para criar a impressão de volume por camadas. Os módulos tem forma de um triângulo, para simular um pássaro de asas abertas.

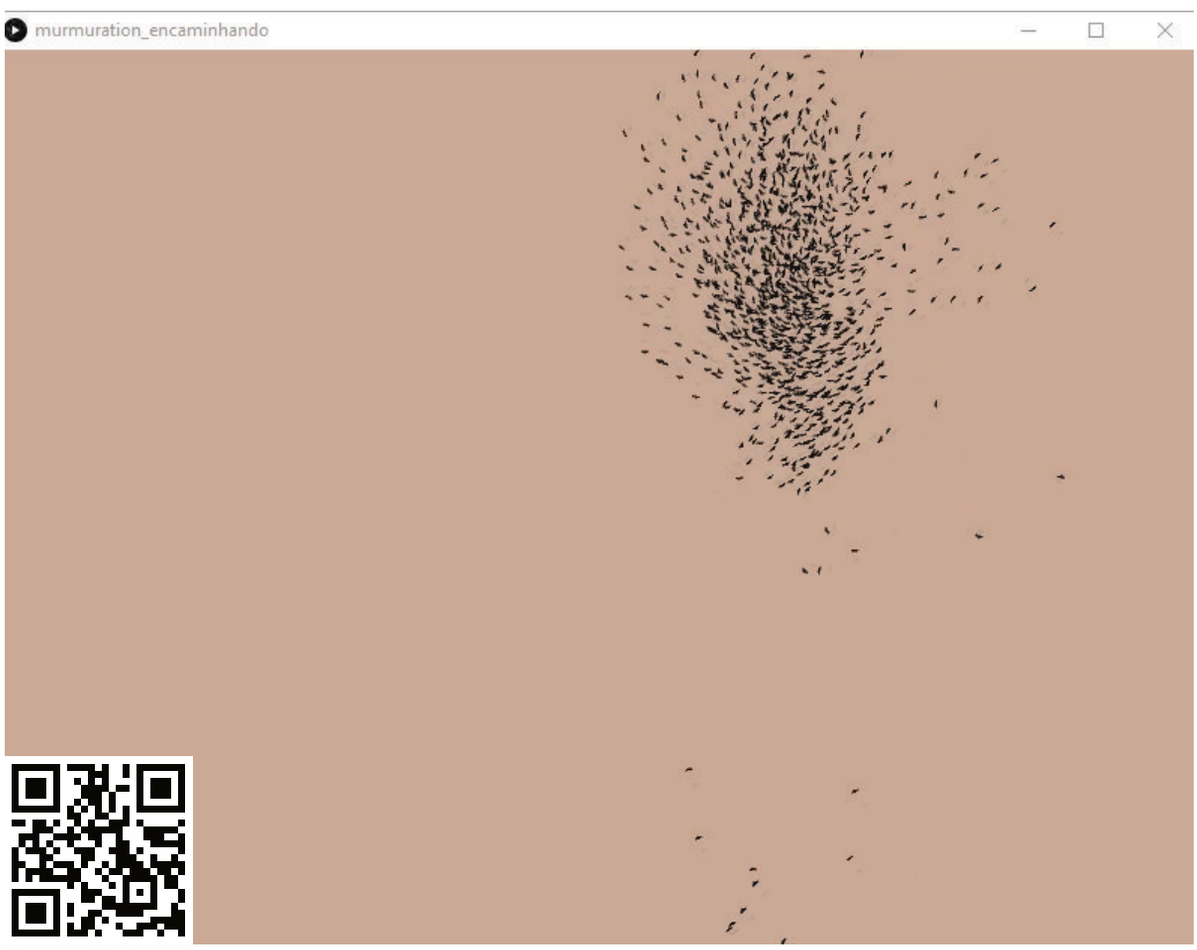
Figura 24. Protótipo de comportamento com atrator na posição do mouse.



Fonte: Elaborado pelo autor, 2019.

O comportamento atingido com o protótipo é próximo o suficiente ao objetivo para o programa final, precisando apenas de ajustes de valores numéricos e a automação de movimento do objeto atrator. O objeto atrator passa a se mover com base em uma função de ruído[1], que gera uma sequência aleatória de números onde cada número é restrito a um valor similar ao valor anterior da sequência, o que torna o movimento mais natural em sua aparência, onde uma função puramente aleatória gera saltos em velocidade e posição que não são compatíveis com o comportamento desejado. Além disso foi adicionado mais um grupo separado de indivíduos para aumentar a densidade de indivíduos no programa e adicionada uma cor de fundo ao programa (figura 25).

Figura 25. Programa Murmúrio de Estorninhos.



Fonte: Elaborado pelo autor, 2019.

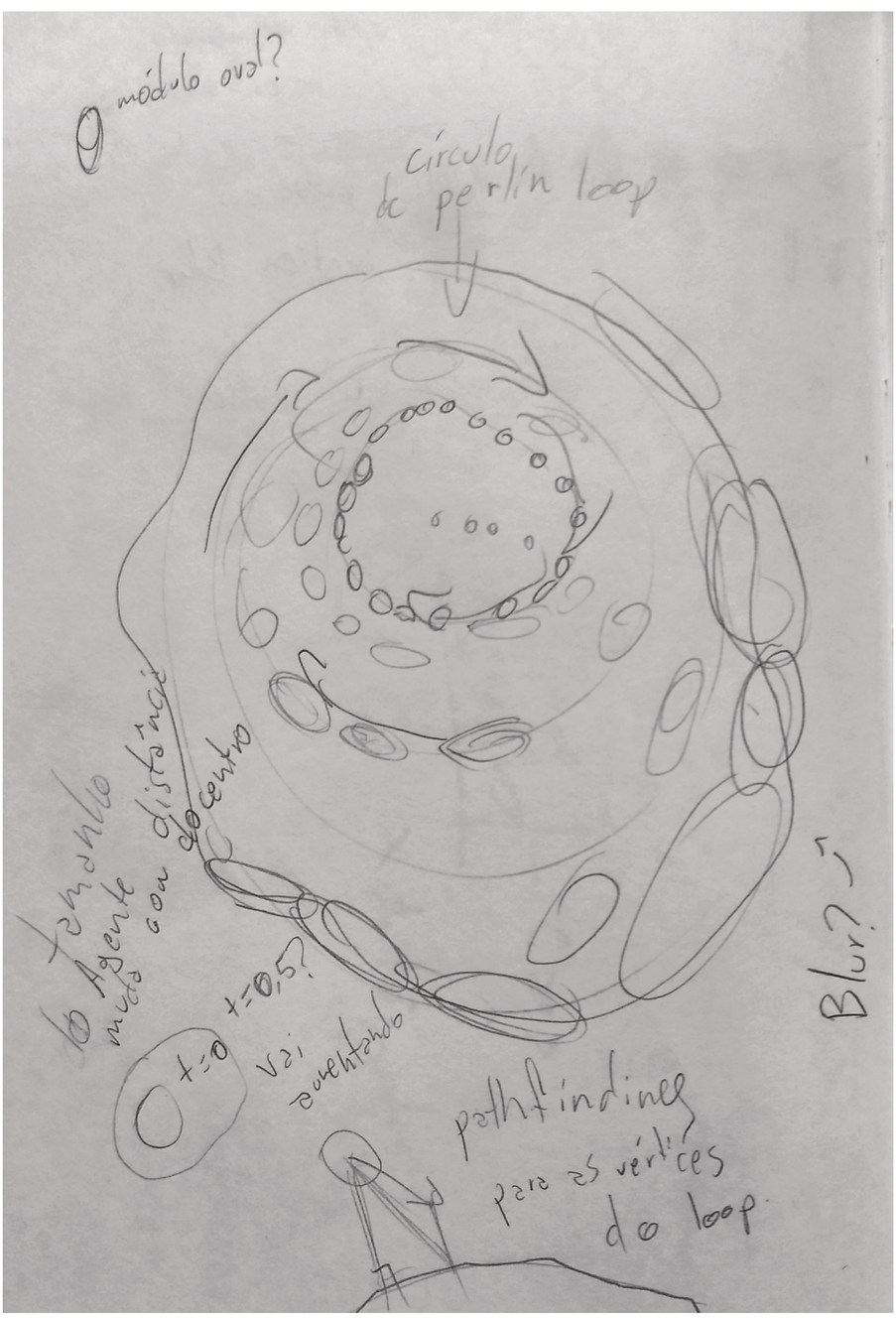
Neste Programa o principal foco foi a criação de um comportamento que remetesse ao comportamento natural através da animação. O programa foi otimizado para ser aplicado como tratamento de superfície por projeção digital de animação, porém também suporta a possibilidade de extrair quadros para impressão por meios tradicionais.

1. Ruído: Descrito por Perlin(1999) como um “primitivo de texturização” usado para criar uma grande variedade de texturas com aparência natural. Possibilita ao programa gerar valores (números, cores, sons ou outros) de forma pseudoaleatória, de modo que uma sequência gerada pela função pareça mais natural — com um número limitando o próximo — evitando os picos de uma sequência aleatória.

4.5.4. Cardume de Peixes em Formação Defensiva

Para simular o movimento circular observado, foi criado um esboço (figura 26) que define algumas possíveis saídas para escrever em código esta simulação. A ideia inicial utiliza um gerador de ruído pseudoaleatório para definir a direção geral do grupo em um ciclo fechado, e uma função que altera o tamanho de cada módulo com relação à distância deste para com o centro do circuito. Nesta fase ainda não foi esboçada uma possibilidade para simular a luz, que reflete em apenas parte dos indivíduos do cardume.

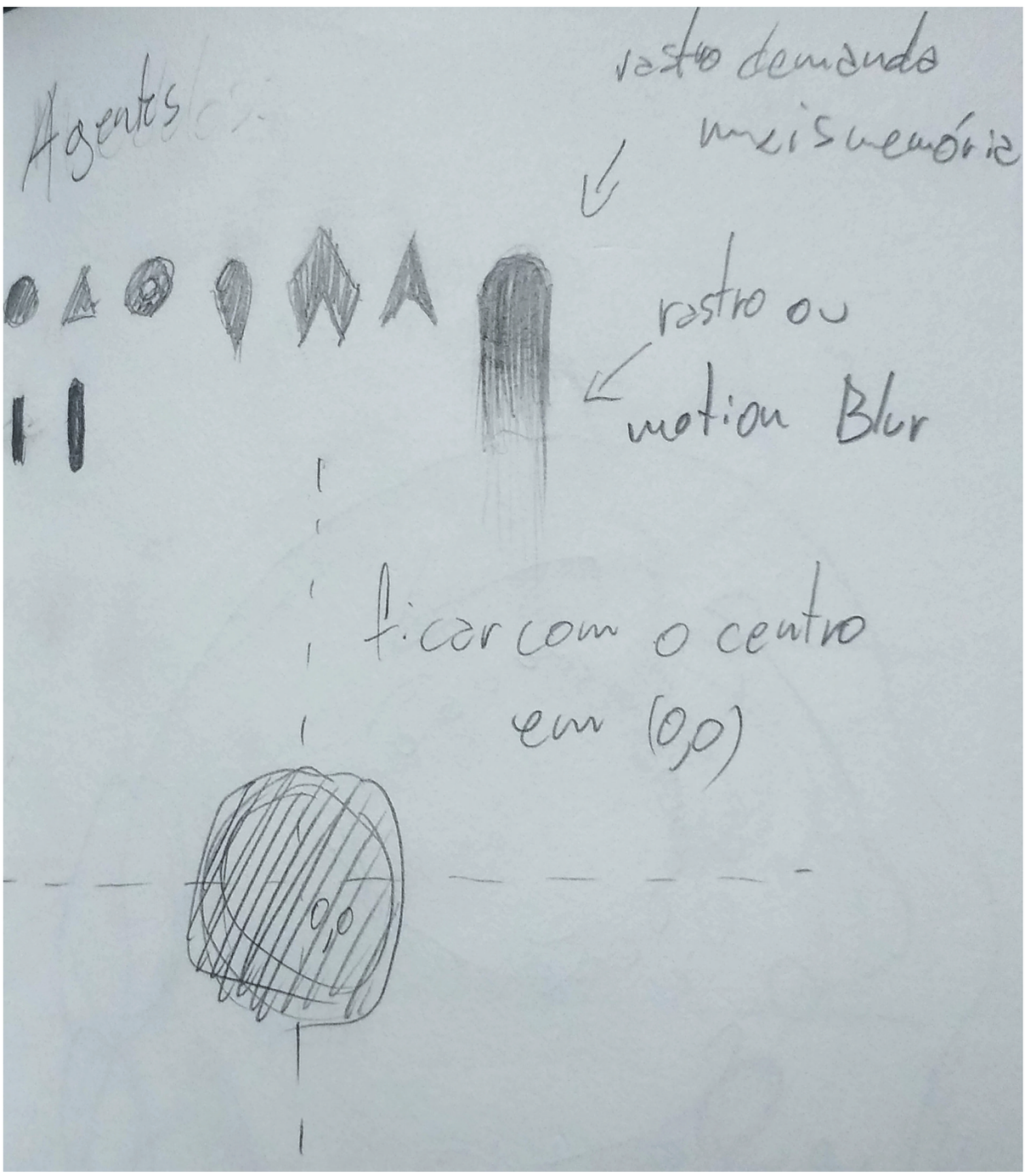
Figura 26. Simulação Cardume.



Fonte: Elaborado pelo autor, 2019.

Foi também esboçado um conjunto de possíveis módulos a serem usados neste ou em outros programas (figura 27).

Figura 27. Agentes geométricos básicos para uso no sistema generativo.



Fonte: Elaborado pelo autor, 2019.

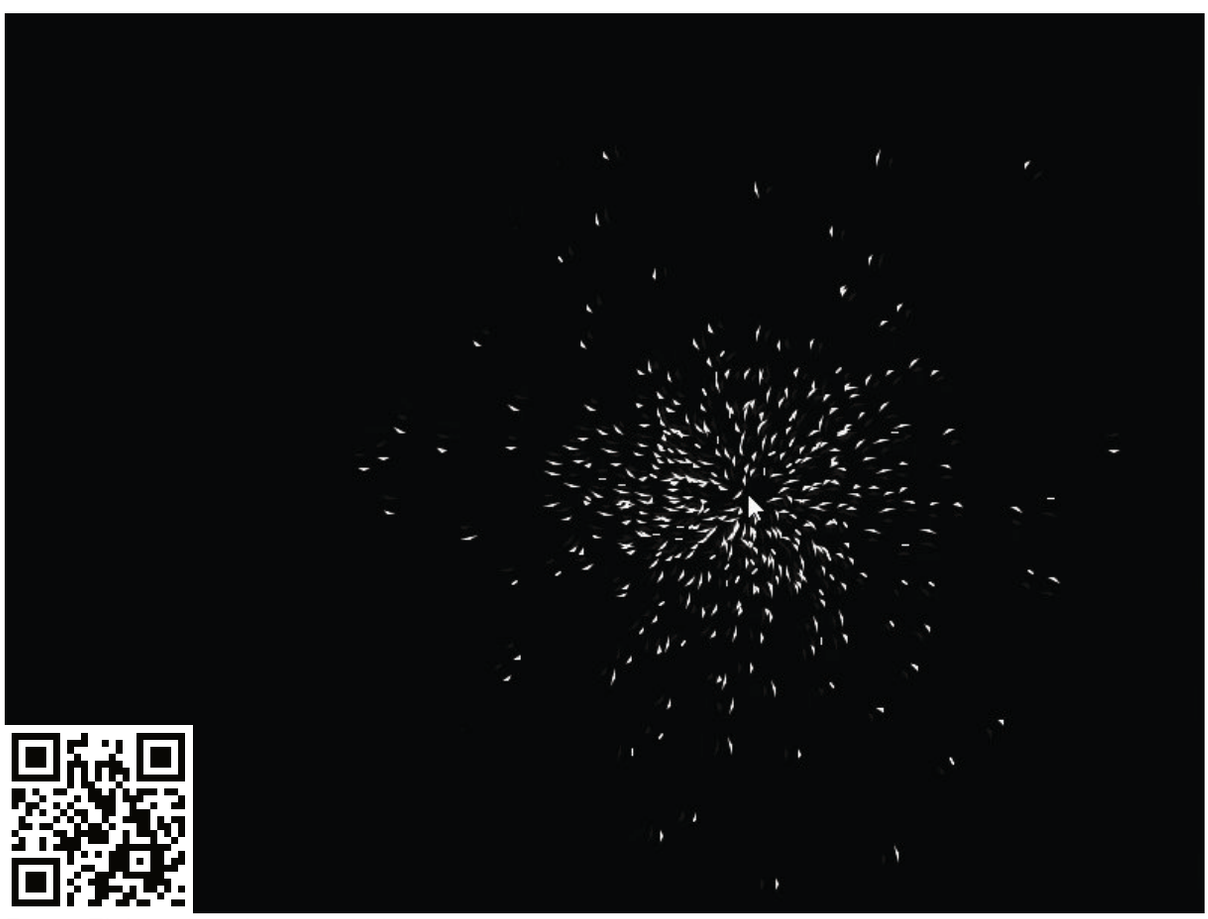
Com base no exemplo estudado do comportamento, foi criada uma paleta de cores para o programa final. Duas cores foram decididas para os módulos e uma para o fundo (figura 28) com base no painel semântico Comportamento e Movimento.

Figura 28. Paleta de cores Cardume

Fonte: Elaborado pelo autor, 2019.

Após escrito, o protótipo criado para o programa Murmúrio de Estorninhos apresentou um comportamento similar ao esboçado. Ao manter o mouse ,e portanto o ponto atrator, parado no mesmo lugar os indivíduos entram em uma formação espiral ao redor do ponto atrator (figura 29). Isso se dá por que o comportamento básico de alinhamento de movimento e as forças de atração ao ponto atrator se equilibram a ponto de forçar uma espécie de órbita.

Figura 29. Comportamento espiral.

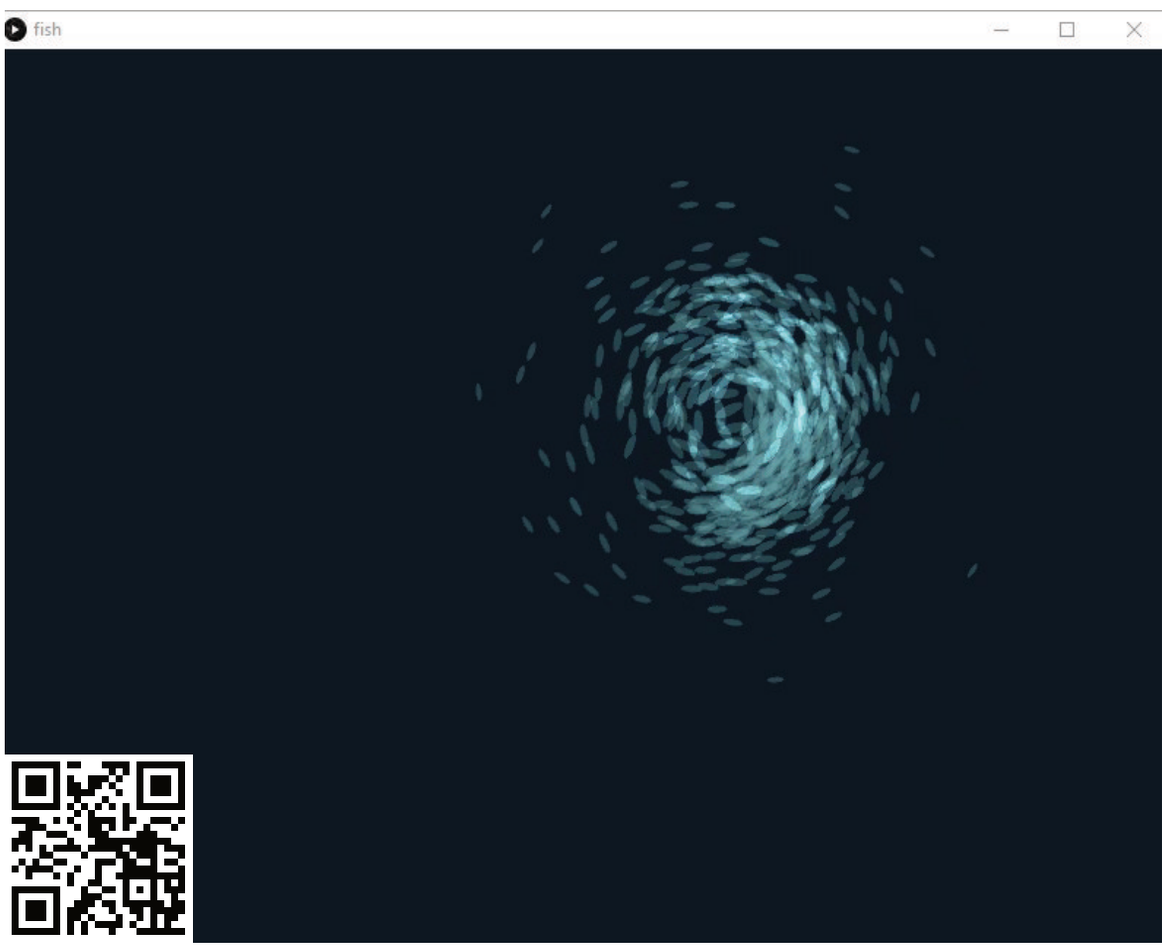


Fonte: Elaborado pelo autor, 2019.

Assim, com base no protótipo existente, foram feitas adaptações para transformar o comportamento do protótipo no comportamento do cardume. Os módulos foram transformados em formas ovais, a velocidade dos módulos for alterada, assim como a força do ponto atrator. O ponto atrator foi transferido do mouse para um ponto móvel, assim como no programa Murmúrio de Estorninhos, porém a fvelocidade de movimento do ponto é menor, para manter o efeito espiral no grupo de indivíduos. O ponto atrator também passa a se manter em uma área mais central, sem a possibilidade de cruzar as laterais do programa.

Foi criado uma função que altera o tamanho do módulo e a cor deste conforme a distância do módulo e o ponto atrator, de forma que quanto mais distante do ponto atrator, menor e mais escuro o módulo. Foram criados três grupos de indivíduos, novamente com a intenção de criar um aspecto de profundidade, e em um destes foi aplicado um filtro de mesclagem nativo da linguagem processing, chamado **ADD.** Este filtro soma o valor de brilho da cor já presente na tela com a cor aplicada, o que dá um efeito de brilho aos módulos quando estes se sobrepõem. A figura 30 ilustra o programa após as alterações de comportamento e as alterações estéticas.

Figura 30. Programa Cardume de Peixes em Formação Defensiva.



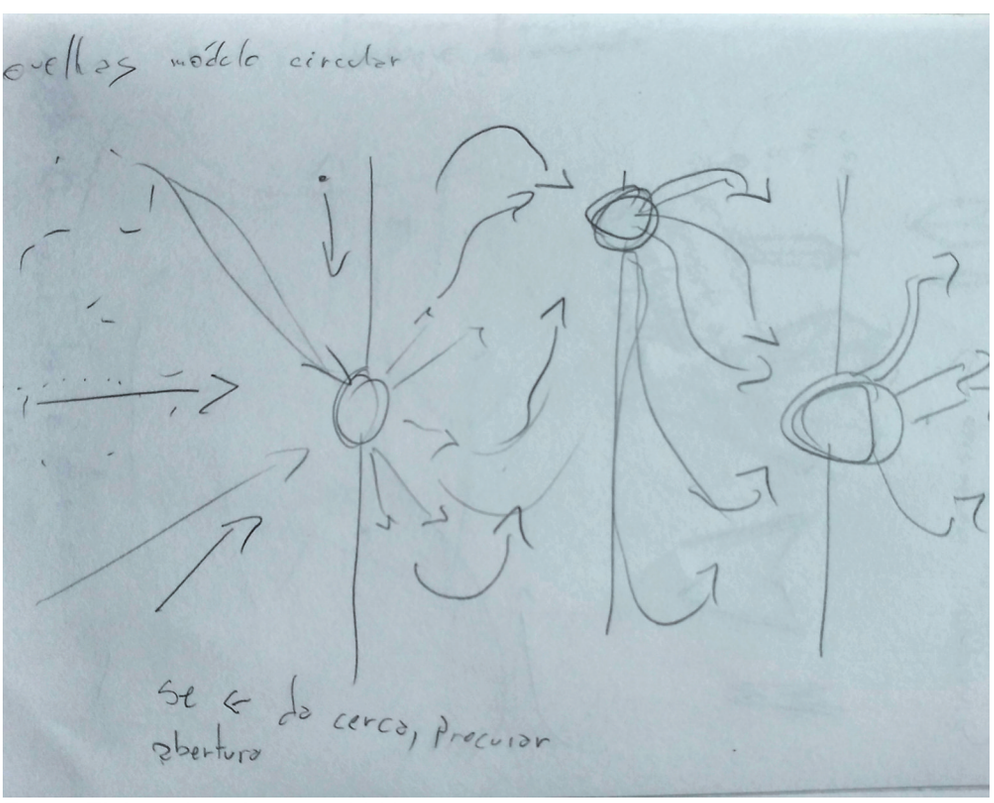
Fonte: Elaborado pelo autor, 2019.

Como nos outros programas, é possível aplicar o resultado através de projeção digital. Neste caso a extração de quadros do programa possibilita a utilização de vários quadros em conjunto, considerando que não existem elementos nas margens da imagem.

4.5.5. Rebanho de Ovelhas Entrando em Cercado

Assim como nos programa anterior, foi primeiro criado um esboço, para estabelecer uma composição básica, para guiar o que precisa ser escrito em código. Para este programa foi definido que par simular o comportamento será necessário criar um ponto onde os módulos se aglomerem, para dar a impressão de aglomeração ao redor de uma porteira de um cercado.

Figura 31. Simulação Rebanho + Cercas.

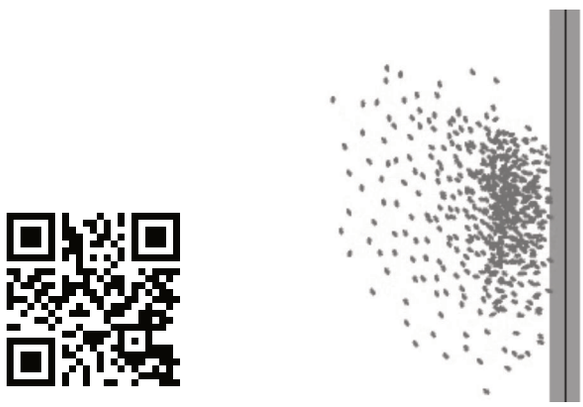


Fonte: Elaborado pelo autor, 2019.

Para este programa se fez necessária a criação de um sistema de colisões para simular as cercas presentes no exemplo observado. Para isso foi usado um cálculo de que mede a projeção escalar do módulo e a posição futura deste (calculada com base em sua velocidade e direção atuais) sobre a cerca (que é uma linha). Deste cálculo é possível encontrar o ponto ao longo da cerca que tenha a menor distância entre a posição futura do módulo (para onde ele está indo) e a linha da cerca. Com esse ponto é possível ordenar ao módulo que ele tome uma direção oposta caso esta

distância seja menor que a espessura da cerca (figura 32).

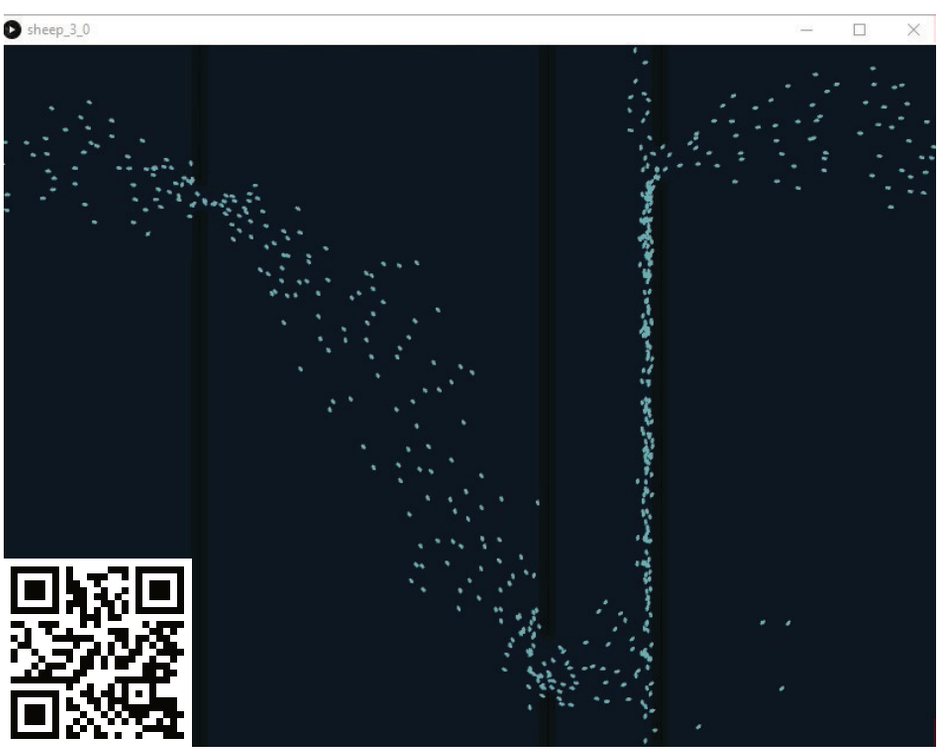
Figura 32. Comportamento de colisão com cerca.



Fonte: Elaborado pelo autor, 2019.

Com o comportamento de colisão escrito, este foi implementado aos comportamentos básicos. Foi também criada uma função que cria três cercas na tela em posições aleatórias, cada uma com uma abertura ao longo de sua extensão, onde os indivíduos conseguem cruzar. Após a criação das cercas, foram criados pontos atratores localizados em cada uma das aberturas das cercas. Em um primeiro momento estes pontos atraíam os indivíduos localizados à esquerda de cada um deles.

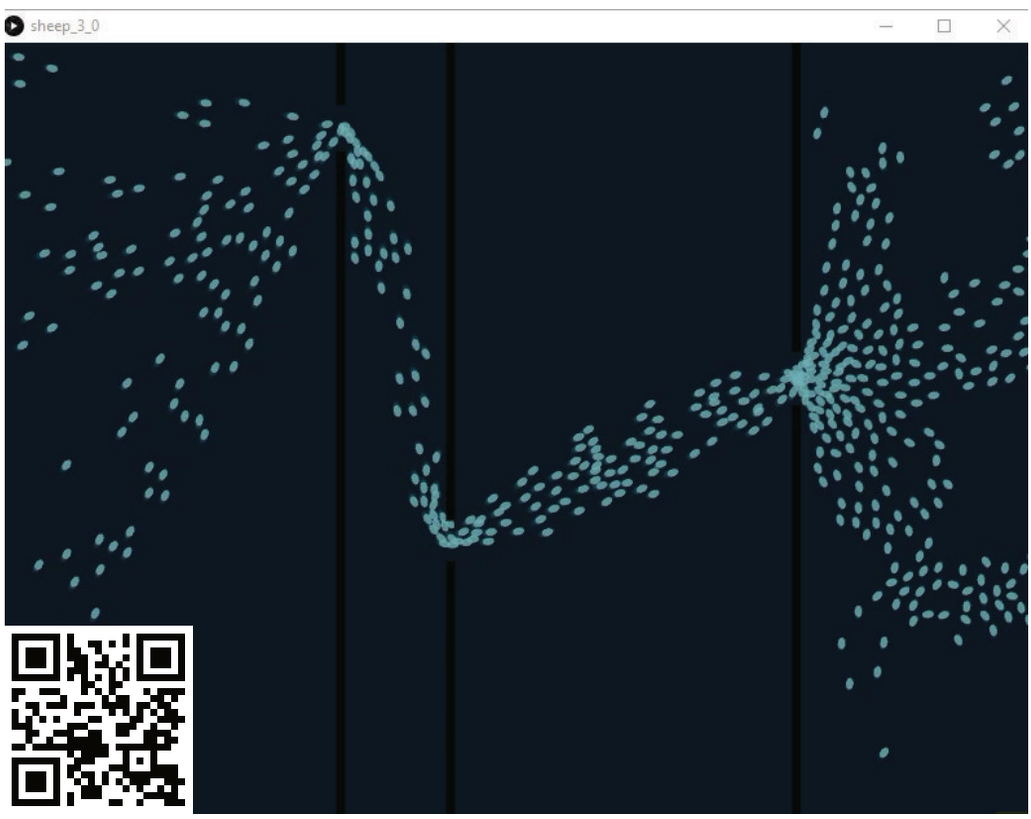
Figura 33. Cercas, aberturas e atratores.



Fonte: Elaborado pelo autor, 2019.

Este modelo, no entanto, possuía um problema onde os indivíduos poderiam sofrer influências de mais de um atrator ao mesmo tempo. Foi então definido que cada módulo seleciona o seu atrator com base em sua posição atual: se o módulo se encontra à esquerda do primeiro atrator, o módulo seleciona o primeiro atrator, se o módulo se encontra entre o primeiro e o segundo, ele seleciona o segundo, e se ele se encontra entre o segundo e o terceiro, o módulo seleciona o terceiro atrator e por fim, se o módulo se encontra à direita do terceiro atrator ele não é atraído por nada, o que eventualmente o enviará para margem direita da tela, onde ele será mandado para o início da tela novamente (figura 34).

Figura 34. Comportamento de atratores corrigido.



Fonte: Elaborado pelo autor, 2019.

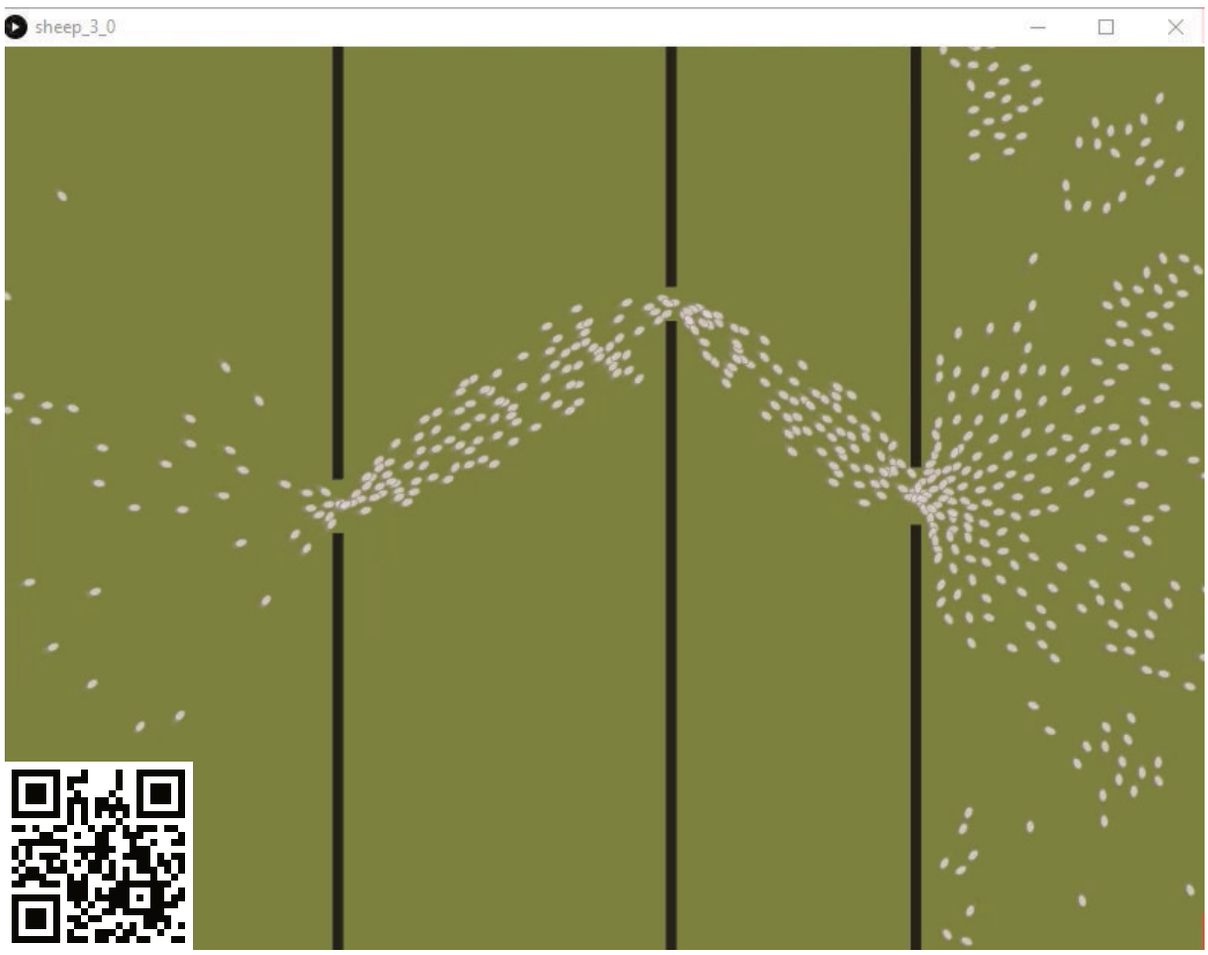
Por fim, foram escolhidas cores que remetam ao comportamento observado (figura 35) e ao painel semântico Comportamento e Movimento. Estas cores foram por fim aplicadas no programa final (figura 36).

Figura 35. Paleta de cores para Rebanho de ovelhas.

Cor dos Módulos Cor de fundo Cor das cercas

Fonte: Elaborado pelo autor, 2019.

Figura 36. Programa Rebanho de Ovelhas Entrando em Cercado.



Fonte: Elaborado pelo artista, 2019.

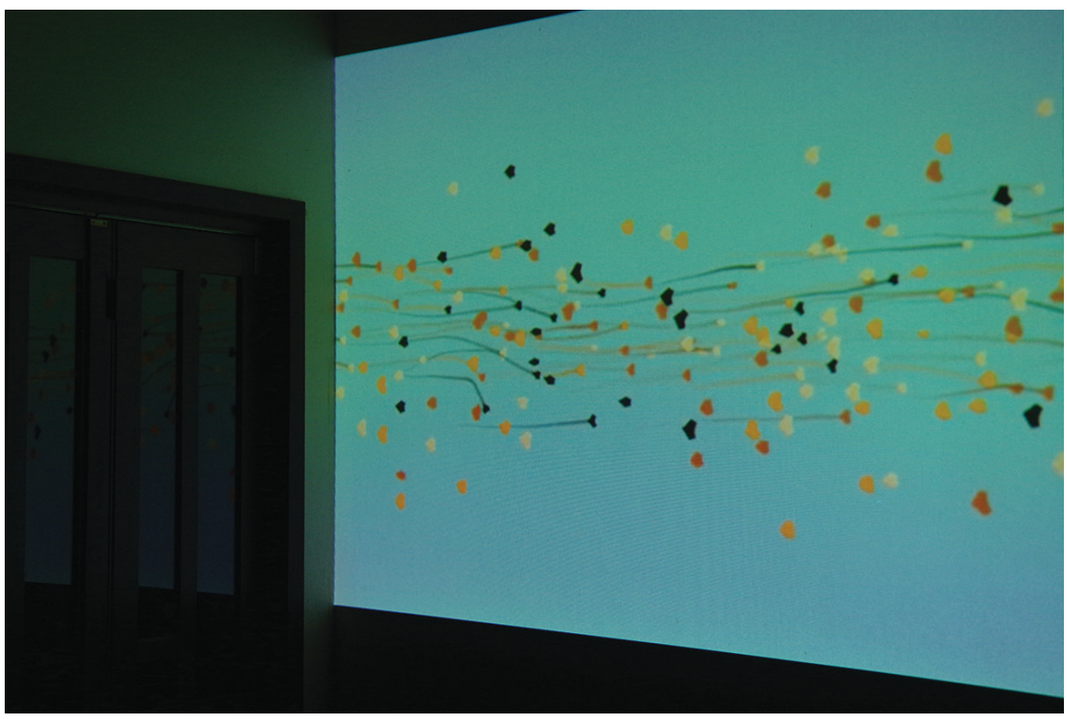
O programa final apresenta uma boa apresentação do comportamento de atratores, porém apresenta também um espaço para visualizar o comportamento base, com as ovelhas virtuais tendo um espaço para se comportar de forma menos guiada.

4.6. FINALIZAÇÃO E RESULTADOS

Ao final do processo de programação obteve-se como resultado um conjunto de quatro programas generativos de imagem: *Borboletas em Migração, Murmúrio de Estorninhos, Cardume de Peixes em Formação Defensiva* e *Rebanho de Ovelhas Entrando em Cercado.* Estes programas foram desenvolvidos para serem aplicados através de projeção digital sobre superfície, mas possibilitar a extração de imagens para impressão tradicional, conforme os requisitos estabelecidos. Foi desenvolvida uma função para capturar o estado do programa ao clique do mouse, e este estado é então salvo como uma imagem, que por sua vez pode ser usada como base para um raport ou ainda ser adulterada em softwares de imagem tradicionais.

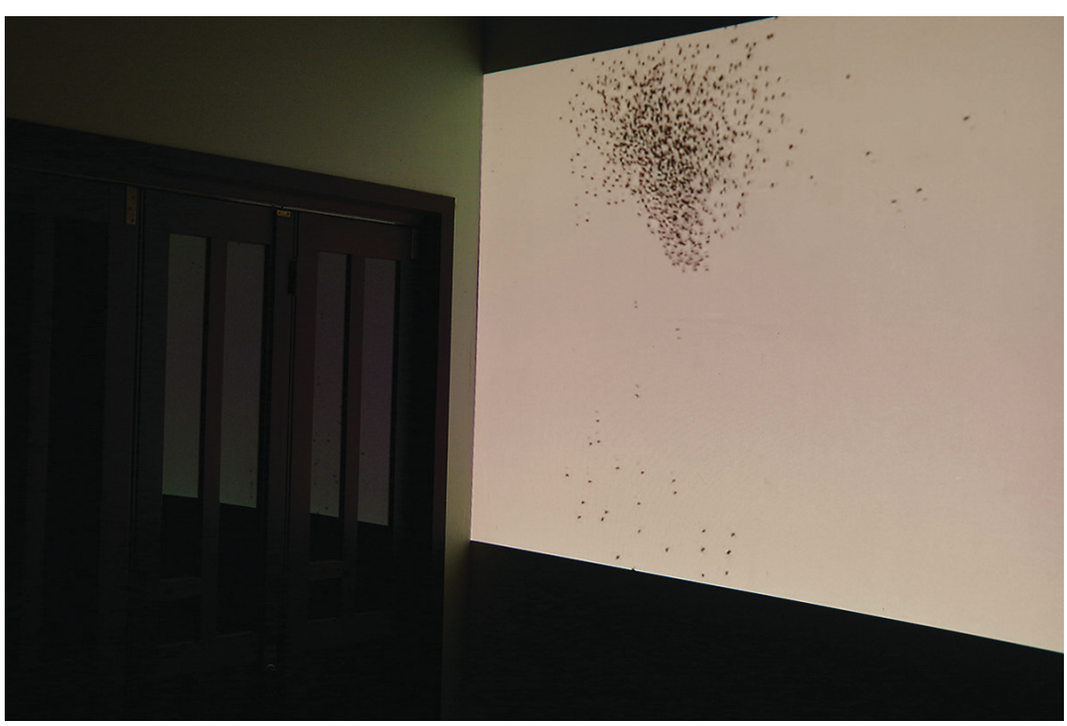
Com a pesquisa finalizada foram aplicados os programas em um cenário de projeção sobre superfície (figuras 37, 38, 39), assim como criado um padrão estático utilizando imagens extraídas de cada um dos programas criados (figuras 40, 41, 42).

Figura 37. Projeção do programa Borboletas em Migração



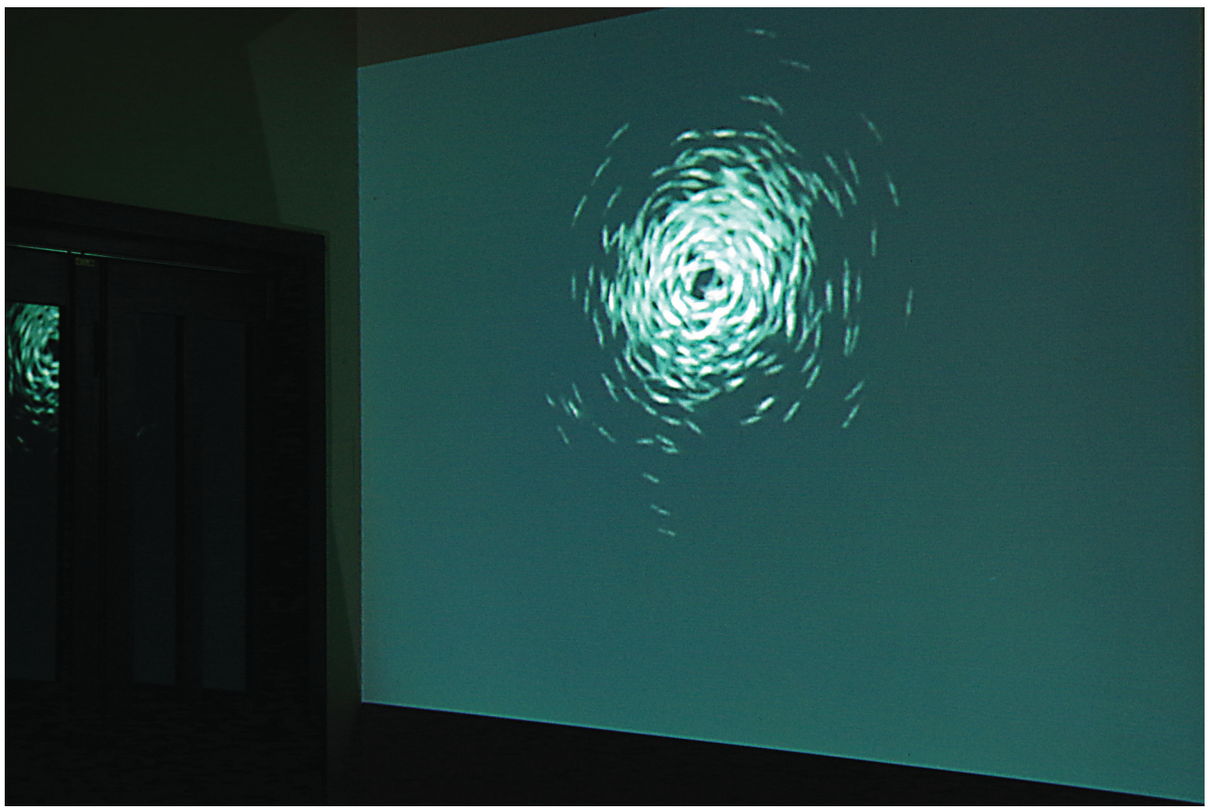
Fonte: Elaborado pelo autor, 2019.

Figura 38. Projeção do programa Murmúrio de Estorninhos.



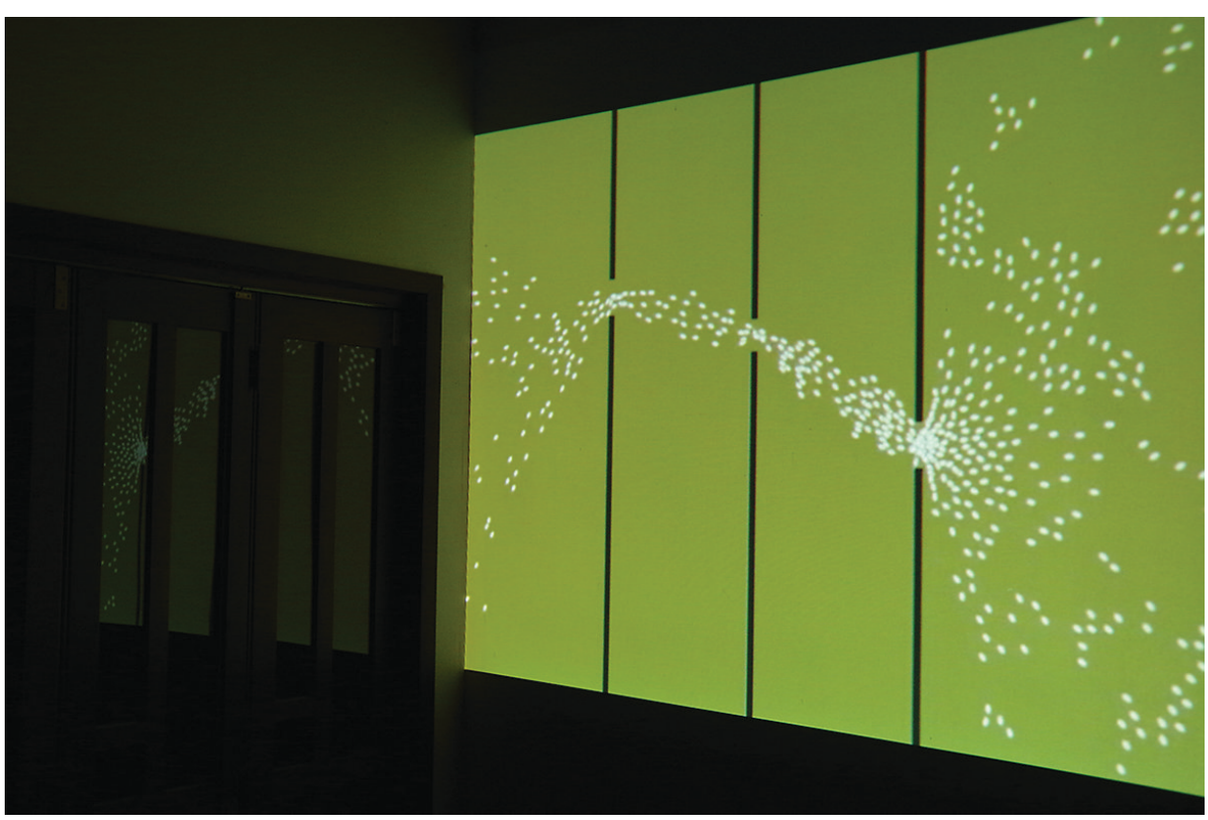
Fonte: Elaborado pelo autor, 2019.

Figura 39. Projeção do programa Cardume de Peixes em Formação Defensiva.



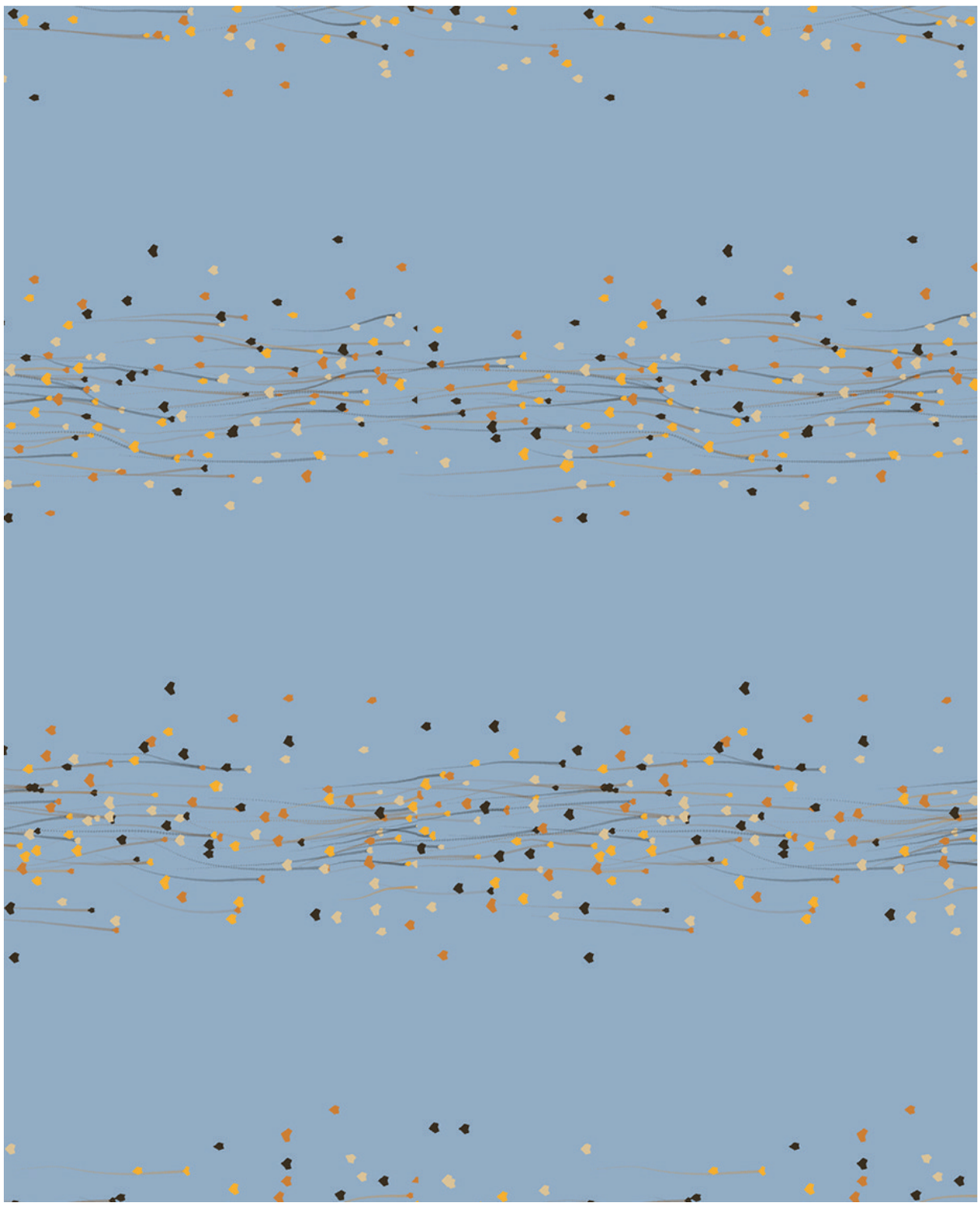
Fonte: Elaborado pelo autor, 2019.

Figura 40. Projeção do programa Rebanho de Ovelhas Entrando em Cercado.



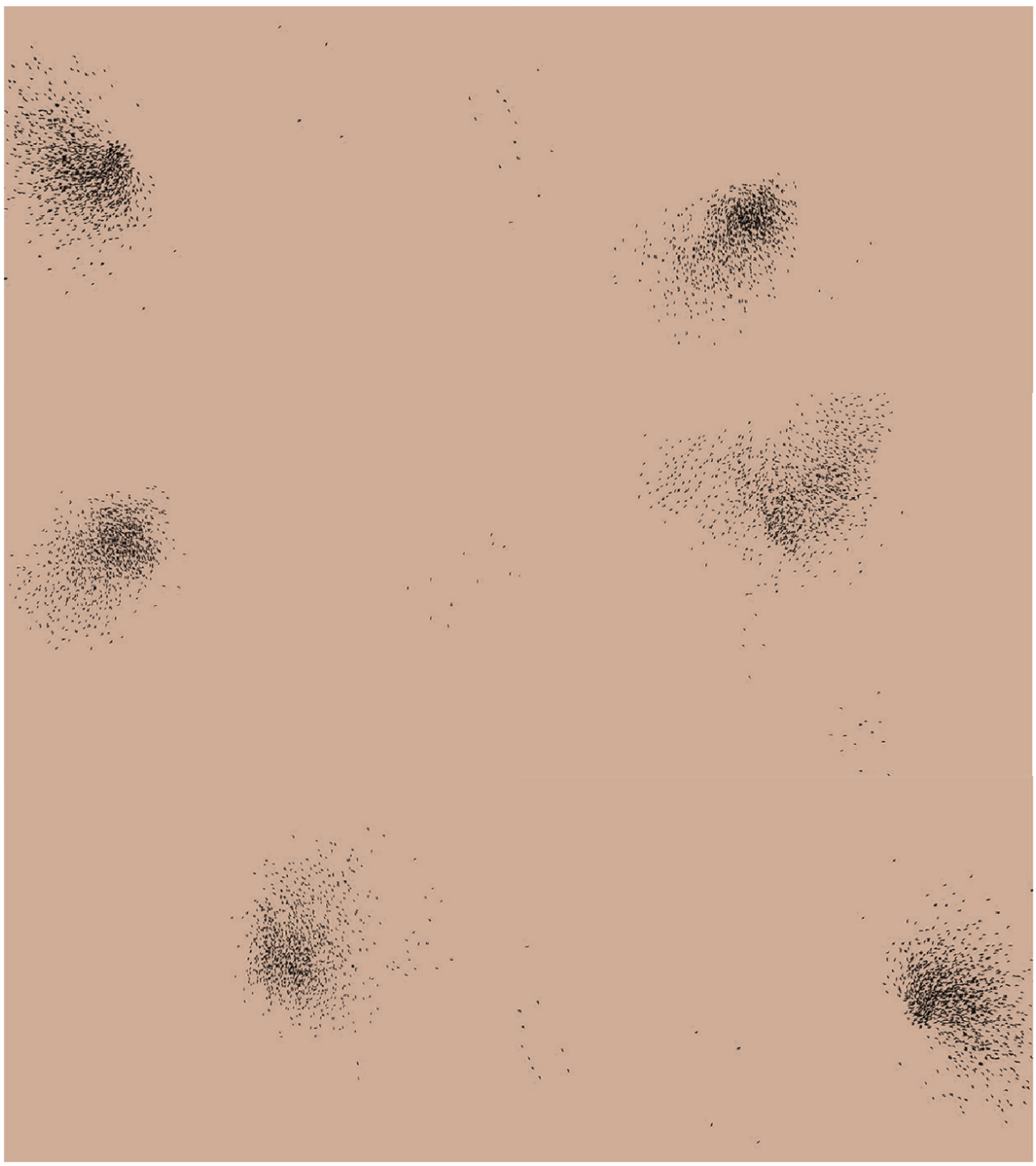
Fonte: Elaborado pelo autor, 2019.

Figura 41. Padrão criado a partir do programa Borboletas em Migração.



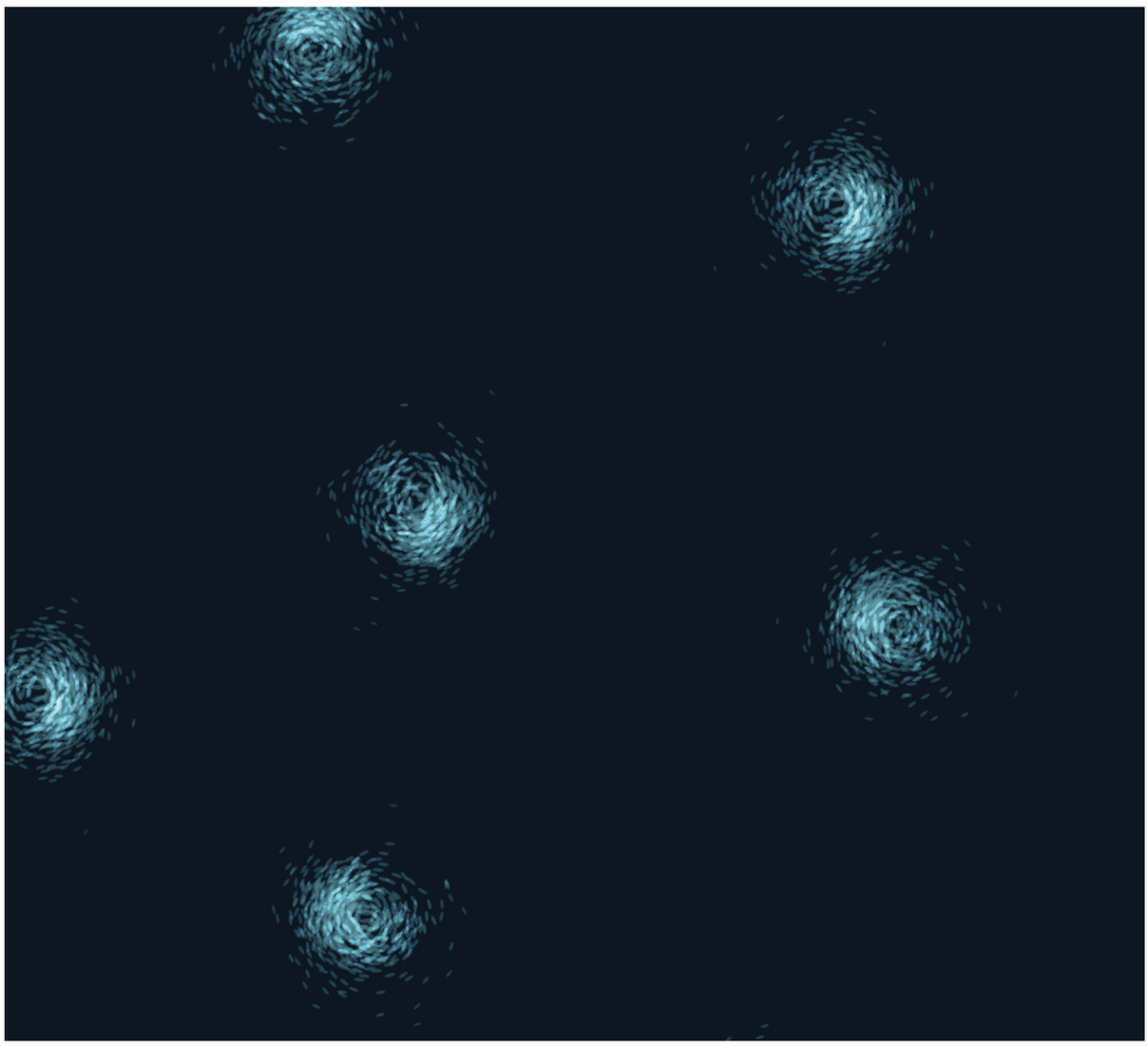
Fonte: Elaborado pelo autor, 2019.

Figura 42. Padrão criado a partir do programa Murmúrio de Estorninhos.



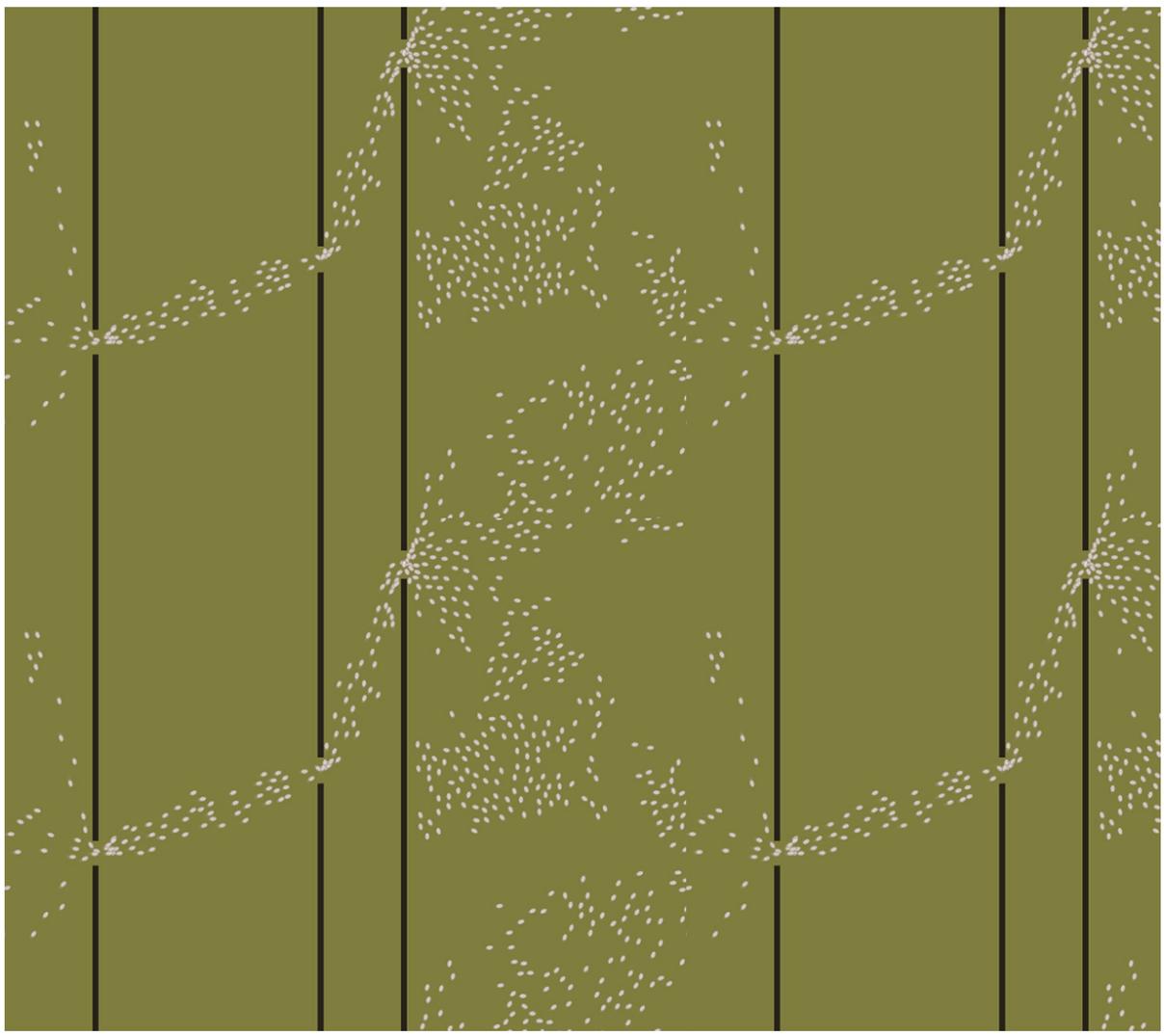
Fonte: Elaborado pelo autor, 2019.

Figura 43. Padrão criado a partir do programa Cardume de Peixes em Formação Defensiva



Fonte: Elaborado pelo autor, 2019.

Figura 44. Padrão criado a partir do programa Rebanho de Ovelhas Entrando em Cercado.



Fonte: Elaborado pelo autor, 2019.

Dos requisitos definidos para o projeto, os que foram atingidos de forma mais bem-sucedida são os que dizem respeito à inspiração em comportamentos naturais e ao modo de como os programas foram criados: programados em processing, utilizando a abordagem de agentes e onde a complexidade se dá pelas interações entre elementos do programa.

5. CONSIDERAÇÕES FINAIS

Este estudo em design de superfície partiu do interesse de abordar técnicas de programação voltadas para fins estéticos. Foi possível perceber também uma relação pouco explorada entre programas generativos e design de superfície. Com isso foi definida a intenção de aliar a programação e a criação de sistemas generativos ao design de superfície, a fim de explorar alguns dos resultados possíveis desta relação.

O objetivo geral, de estabelecer uma base de conhecimento sobre processos generativos para criar um conjunto de sistemas generativos de imagem para uso em superfícies, foi estabelecido para guiar a pesquisa como um todo, e foi amparado pelos objetivos específicos, que orientaram as etapas do processo.

A pesquisa sobre a base do que é considerado design de superfície serviu para identificar elementos que caracterizam e conceituam o campo. A partir disto a contextualização de novas técnicas e tecnologias associadas ao design de superfície serviu de apoio para caracterizar os processos generativos e as aplicações alternativas como merecedoras de atenção dentro do campo.

Apresentou-se a necessidade de compreender os processos e os sistemas generativos de forma exaustiva, para criar uma definição nítida e clara sobre o que eles são e como funcionam. Para isto fez-se uma pesquisa de definições existentes, desde como é conceituada a atividade generativa e o que caracteriza esta atividade, até conceitos que são frequentemente associados à atividade e ao design generativo, como algoritmo, emergência e complexidade.

Com o conhecimento base sobre design de superfície e possibilidades dentro do campo assim como um conceito sólido para o que são e como funcionam sistemas generativos foi possível definir uma metodologia compatível com as etapas de programação necessárias para o desenvolvimento dos programas. A metodologia foi construída tomando as etapas de Bonsiepe (1984), Baxter (2017) e Valério (2013) que pareceram mais pertinentes para guiar o rumo do projeto.

Na primeira fase de preparação, a Problematização serve criar um ponto de partida para o projeto, estabelecendo uma direção inicial. As perguntas “o quê, por quê e como” serviram também para estabelecer limites para o trabalho.

No segundo momento, as análises elaboradas serviram para melhor compreender as formas que o design generativo toma diante ao mercado, bem como a evolução do uso de sistemas generativos para criação imagética. As análises serviram também para definir e confirmar a linguagem de programação a ser utilizada para criar os programas, considerando principalmente a disponibilidade de recursos de aprendizado para cada linguagem considerada.

Ainda dentro da etapa de análises, foram escolhidos e analisados os

comportamentos naturais que serviram de inspiração para os programas finais. Esta análise serviu para inspirar a direção estética de cada programa, mas principalmente como traduzir os comportamentos e movimentos dos animais em lógica de programação.

Deste modo, os pontos apresentados suportaram a criação dos quatro programas generativos para uso em superfícies: *Borboletas em Migração, Murmúrio de Estorninhos, Cardume de Peixes em Formação Defensiva* e *Rebanho de Ovelhas Entrando em Cercado.* Assim, é possível afirmar que o objetivo geral estabelecido foi alcançado com êxito, considerando a vontade inicial de explorar um modo diferente de criar designs de superfície, e abrir a possibilidade de usar as capacidades computacionais existentes de de formas alternativas aos softwares usuais, principalmente levando em consideração o que normalmente é explorado no curso de Especialização em Design de Superfície da Universidade Federal de Santa Maria.

Embora a pesquisa apresentada esteja finalizada diante da proposta deste trabalho, ainda existem possibilidades inexploradas dentro dos assuntos e abordagens compreendidas.

REFERÊNCIAS

BAXTER, M. **Product Design:** A practical guide to systematic methods of new product development. Boca Raton: CRC Press, 2017.

BENSE, M. **Pequena Estética.** São Paulo: Perspectiva, 2009.

BONSIEPE, G. (Coord.). **Metodologia Experimental**: Desenho industrial. Brasília: CNPq/Coordenação Editorial, 1984.

CORMEN, T. H. et al. **Introduction to Algorithms.** Cambridge, Massachusetts: MIT Press, 2003.

DAY, A. *When Skyscrapers Are Your Screen*. ***New York Times***, Nova Iorque, 2012. Disponível em: <https://www.nytimes.com/2012/01/29/arts/design/video-mapping-artists-use-light-as-a-medium.html?_r=1&>. Acesso em: 16 jul. 2019.

GALANTER, P. ***What is Generative Art?*** *Complexity Theory as a Context for Art Theory.* In: 6th Generative Art Conference Proceedings(2003). Milão, 2003. Disponível em: <http://philipgalanter.com/downloads/ga2003_what_is_genart.pdf>. Acesso em: 7 jun. 2019.

GEORG Nees. **Compart.** Disponível em: <http://dada.compart-bremen.de/item/agent/15>. Acesso em: 30 jul. 2019.

GREENBERG, I. ***Processing***: *Creative Coding and Computational Art*. Berkeley: Apress, 2007.

HOBBS, T. **Code goes in, Art comes out**. Palestra proferida no Clojure Conj 2018, Durham (Carolina do Norte), 2018. Disponível em: <https://www.youtube.com/watch?v=LBpqoj2nOQo>. Acesso em: 29 jul. 2019.

LARANJEIRA, M. A. et al. Design Generativo de Superfícies: uma análise do uso de programação para o desenvolvimento de estamparia. **Modapalavra**, Florianópolis, v. 11, n. 21, p.1-17, 23 nov. 2018. Semestral. Universidade do Estado de Santa Catarina. Disponível em: <http://www.revistas.udesc.br/index.php/modapalavra/article/view/10371/7186>. Acesso em: 16 jul. 2019.

LARANJEIRA, M. A.; MARAR, J. F. **O Uso de Programação no Design de Padrões**, p. 4958-4969 . In: Anais do 13º Congresso Pesquisa e Desenvolvimento em Design (2018). São Paulo: Blucher, 2019.

LUPTON, E.; PHILLIPS, J. C. **Graphic Design The New Basics.** Nova Iorque: Princeton Architectural Press, 2015.

MALIK, A. K. ***Mass Customization!*** *An approach through Generative Design*. 2014. 61 p. Dissertação (Master of Fine Arts) - Department of Design, Virginia Commonwealth University, 2014.

McCORMACK, J.; DORIN, A. **Art, Emergence, and the Computational Sublime.** In: Proceedings of the Second International Conference on Generative Systems in the Electronic Arts (2001). Victoria, Australia: Centre for Electronic Media Art, 2001.

McCORMACK, J.; DORIN, A.; INNOCENT, T. ***Generative design:*** *a paradigm for design research*. In: Redmond, J. et. al. (eds) Proceedings of Futureground(2004). Melbourne: Design Research Society, 2004.

MONRO, G. *Emergence and Generative Art.* **Leonardo,** [S.l.], v. 42, out. 2009. MIT Press. Disponível em: <https://www.researchgate.net/publication/241898283_Emergence_and_Generative-_Art>. Acesso em: 18 jul. 2019.

PEARSON, M. ***Generative Art:*** *A Practical Guide Using Processing.* Nova Iorque: Manning, 2011.

PERLIN, K. ***What is noise.*** Disponível em: <https://web.archive.org/web/20071008162206/http://www.noisemachine.com/talk1/3.html>. Acesso em: 9 nov. 2019.

NOGUCHI, H. **Mozart - Musical Game in C K. 516f*.** 1997. Disponível em: <http://www.asahi-net.or.jp/~rb5h-ngc/e/k516f.htm>. Acesso em: 29 jul. 2019.

REYNOLDS, C. W. **Steering Behaviors for Autonomous Characters**. 1999. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.636&rep=rep1&type=pdf>. Acesso em: 17 out. 2019.

RUBIM, R. **Desenhando a Superfície**. São Paulo: Rosari, 2004.

RUSSELL, A. ***Repeatless***: *transforming surface pattern with generative design*. In: Shapeshifting Conference(2014). Auckland: Auckland University of Technology, 2014.

RÜTHSCHILLING, E. A. **Design de Superfície**. Porto Alegre: Ed. UFRGS, 2008.

SCHWARTZ, A. R. **Design de Superfície:** por uma visão projetual geométrica e tridimensional, 2008. Dissertação (Mestrado em Desenho Industrial) –Universidade Estadual Paulista, Bauru.

VALÉRIO, J. H. **Processing design**: reflexões sobre design gerativo. Dissertação (Mestrado em Educação, Arte e História da Cultura) – Universidade Presbiteriana Mackenzie. São Paulo, 2013

VEROSTKO, R. ***The Algorists.*** Disponível em: <http://www.verostko.com/algorist.html>. Acesso em: 29 jul. 2019.

WHAT IS VJING. **Findwords.** Disponível em: <https://findwords.info/term/vjing>. Acesso em: 8 nov. 2019.

APÊNDICE A - REPOSITÓRIO DE CÓDIGO

O código para os programas generativos pode ser encontrado no repositório público no link: https://github.com/FeliPK/TCC_PGDS_sistemasgen/tree/master

No repositório é possível encontrar os arquivos para edição de código para o ambiente Processing e o próprio sistema do repositório permite visualizar o texto do código no browser.

ANEXO A - VISÃO BÁSICA DO PROCESSING

Processing Overview Processing Foundation

This tutorial is adapted from the book, Visualizing Data by Ben Fry, O'Reilly 2007. © 2007 Ben Fry.

Ben Fry and Casey Reas

Processing is a simple programming environment that was created to make it easier to develop visually oriented applications with an emphasis on animation and providing users with instant feedback through interaction. The developers wanted a means to "sketch" ideas in code. As its capabilities have expanded over the past decade, Processing has come to be used for more advanced production-level work in addition to its sketching role. Originally built as a domain-specific extension to Java targeted towards artists and designers, Processing has evolved into a full-blown design and prototyping tool used for large-scale installation work, motion graphics, and complex data visualization.

Processing is based on Java, but because program elements in Processing are fairly simple, you can learn to use it even if you don't know any Java. If you're familiar with Java, it's best to forget that Processing has anything to do with Java for a while, until you get the hang of how the API works.

The latest version of Processing can be downloaded at http://processing.org/download

An important goal for the project was to make this type of programming accessible to a wider audience. For this reason, Processing is free to download, free to use, and open source. But projects developed using the Processing environment and core libraries can be used for any purpose. This model is identical to GCC, the GNU Compiler Collection. GCC and its associated libraries (e.g. libc) are open source under the GNU Public License (GPL), which stipulates that changes to the code must be made available. However, programs created with GCC (examples too numerous to mention) are not themselves required to be open source.

Processing consists of:

- The Processing Development Environment (PDE). This is the software that runs when you double-click the Processing icon. The PDE is an Integrated Development Environment (IDE) with a minimalist set of features designed as a simple introduction to programming or for testing one-off ideas.
- A collection of functions (also referred to as commands or methods) that make up the "core" programming interface, or API, as well as several libraries that support more advanced features such as sending data over a network, reading live images from a webcam, and saving complex imagery in PDF format.
- A language syntax, identical to Java but with a few modifications.
- An active online community, based at http://processing.org.

For this reason, references to "Processing" can be somewhat ambiguous. Are we

talking about the API, the development environment, or the web site? We'll be careful in this text when referring to each.

Sketching with Processing

A Processing program is called a sketch. The idea is to make Java-style programming feel more like scripting, and adopt the process of scripting to quickly write code. Sketches are stored in the sketchbook, a folder that's used as the default location for saving all of your projects. Sketches that are stored in the sketchbook can be accessed from File → Sketchbook. Alternatively, File → Open... can be used to open a sketch from elsewhere on the system.

Advanced programmers need not use the PDE, and may instead choose to use its libraries with the Java environment of choice. However, if you're just getting started, it's recommended that you use the PDE for your first few projects to gain familiarity with the way things are done. While Processing is based on Java, it was never meant to be a Java IDE with training wheels. To better address our target audience, the conceptual model (how programs work, how interfaces are built, and how files are handled) is somewhat different from Java.

Hello world

The Processing equivalent of a "Hello World" program is simply to draw a line:

```
line(15, 25, 70, 90);
```

Enter this example and press the Run button, which is an icon that looks like the Play button from any audio or video device. Your code will appear in a new window, with a gray background and a black line from coordinate (15, 25) to (70, 90). The (0, 0) coordinate is the upper left-hand corner of the display window. Building on this program to change the size of the display window and set the background color, type in the code below:

```
size(400, 400);
background(192, 64, 0);
stroke(255);
line(150, 25, 270, 350);
```

This version sets the window size to 400 x 400 pixels, sets the background to an orange-red, and draws the line in white, by setting the stroke color to 255. By default, colors are specified in the range 0 to 255. Other variations of the parameters to the `stroke()` function provide alternate results:

```
stroke(255);              // sets the stroke color to white
stroke(255, 255, 255);    // identical to the line above
stroke(255, 128, 0);      // bright orange (red 255, green 128, blue 0)
stroke(#FF8000);          // bright orange as a web color
stroke(255, 128, 0, 128); // bright orange with 50% transparency
```

The same alternatives work for the `fill()` function, which sets the fill color, and the `background()` function, which clears the display window. Like all Processing functions that affect drawing properties, the fill and stroke colors affect all geometry drawn to the screen until the next fill and stroke functions.

Hello mouse

A program written as a list of statements (like the previous examples) is called a static sketch. In a static sketch, a series of functions are used to perform tasks or create a single image without any animation or interaction. Interactive programs are drawn as a series of frames, which you can create by adding functions titled `setup()` and `draw()` as shown in the code below. These are built-in functions that are called automatically.

```
void setup() {
  size(400, 400);
  stroke(255);
  background(192, 64, 0);
}

void draw() {
  line(150, 25, mouseX, mouseY);
}
```

The `setup()` block runs once, and the `draw()` block runs repeatedly. As such, `setup()` can be used for any initialization; in this case, setting the screen size, making the background orange, and setting the stroke color to white. The `draw()` block is used to handle animation. The `size()` function must always be the first line inside `setup()`.

Because the `background()` function is used only once, the screen will fill with lines as the mouse is moved. To draw just a single line that follows the mouse, move the `background()` function to the `draw()` function, which will clear the display window (filling it with orange) each time `draw()` runs.

```
void setup() {
  size(400, 400);
  stroke(255);
}

void draw() {
  background(192, 64, 0);
  line(150, 25, mouseX, mouseY);
}
```

Static programs are most commonly used for extremely simple examples, or for scripts that run in a linear fashion and then exit. For instance, a static program might start, draw a page to a PDF file, and exit.

Most programs will use the `setup()` and `draw()` blocks. More advanced mouse handling can also be introduced; for instance, the `mousePressed()` function will be called whenever the mouse is pressed. In the following example, when the mouse is pressed, the screen is cleared via the `background()` function:

```
void setup() {
  size(400, 400);
  stroke(255);
}

void draw() {
  line(150, 25, mouseX, mouseY);
}

void mousePressed() {
  background(192, 64, 0);
}
```

Exporting and distributing your work

One of the most significant features of the Processing environment is its ability to bundle your sketch into an application with just one click. Select File → Export Application to package your current sketch as an application. This will bundle your sketch as an application for Windows, Mac OS X, or Linux depending on which operating system you're exporting from. The application folders are overwritten whenever you export—make a copy or remove them from the sketch folder before making changes to the contents of the folder. Alternatively, you can turn off the automatic file erasure in the Preferences.

More about the export features can be found in the reference at
http://processing.org/reference/environment/#Export

Creating images from your work

If you don't want to distribute the actual project, you might want to create images of its output instead. Images are saved with the `saveFrame()` function. Adding `saveFrame()` at the end of `draw()` will produce a numbered sequence of TIFF-format images of the program's output, named screen-0001.tif, screen-0002.tif, and so on. A new file will be saved each time `draw()` runs —watch out, this can quickly fill your sketch folder with hundreds of files. You can also specify your own name and file type for the file to be saved with a function like:

```
saveFrame("output.png")
```

To do the same for a numbered sequence, use # (hash marks) where the numbers should be placed:

```
saveFrame("output-####.png");
```

For high quality output, you can write geometry to PDF files instead of the screen, as described in the later section about the `size()` function.

Examples and reference

While many programmers learn to code in school, others teach themselves and learn on their own. Learning on your own involves looking at lots of other code: running, altering, breaking, and enhancing it until you can reshape it into something new. With this learning model in mind, the Processing software download includes hundreds of examples that demonstrate different features of the environment and API.

The examples can be accessed from the File → Examples menu. They're grouped into categories based on their function (such as Motion, Typography, and Image) or the libraries they use (PDF, Network, and Video).

Find an interesting topic in the list and try an example. You'll see functions that are familiar, e.g. `stroke()`, `line()`, and `background()`, as well as others that have not yet been covered. To see how a function works, select its name, and then right-click and choose Find in Reference from the pop-up menu (Find in Reference can also be found beneath the Help menu). This will open the reference for that function in your default web browser.

In addition to a description of the function's syntax, each reference page includes an example that uses the function. The reference examples are much shorter (usually

four or five lines apiece) and easier to follow than the longer code examples.

More about size()

The `size()` function sets the global variables width and height. For objects whose size is dependent on the screen, always use the width and height variables instead of a number. This prevents problems when the size() line is altered.

```
size(400, 400);

// The wrong way to specify the middle of the screen
ellipse(200, 200, 50, 50);

// Always the middle, no matter how the size() line changes
ellipse(width/2, height/2, 50, 50);
```

In the earlier examples, the `size()` function specified only a width and height for the window to be created. An optional parameter to the `size()` function specifies how graphics are rendered. A renderer handles how the Processing API is implemented for a particular output function (whether the screen, or a screen driven by a high-end graphics card, or a PDF file). The default renderer does an excellent job with high-quality 2D vector graphics, but at the expense of speed. In particular, working with pixels directly is slow. Several other renderers are included with Processing, each having a unique function. At the risk of getting too far into the specifics, here's a description of the other possible drawing modes to use with Processing.

```
size(400, 400, P2D);
```

The P2D renderer uses OpenGL for faster rendering of two-dimensional graphics, while using Processing's simpler graphics APIs and the Processing development environment's easy application export.

```
size(400, 400, P3D);
```

The P3D renderer also uses OpenGL for faster rendering. It can draw three-dimensional objects and two-dimensional object in space as well as lighting, texture, and materials.

```
size(400, 400, PDF, "output.pdf");
```

The PDF renderer draws all geometry to a file instead of the screen. To use PDF, in addition to altering your size() function, you must select Import Library, then PDF from the Sketch menu. This is a cousin of the default renderer, but instead writes directly to PDF files.

Loading and displaying data

One of the unique aspects of the Processing API is the way files are handled. The `loadImage()` and `loadStrings()` functions each expect to find a file inside a folder named data, which is a subdirectory of the sketch folder.

Advanced Topic: Notes on the data folder

The data folder addresses a common frustration when dealing with code that is tested locally but deployed over the web. Like Java, software written with Processing is

subject to security restrictions that determine how a program can accessresources such as the local hard disk or other serversvia the Internet. This prevents malicious developersfrom writing codethat could harm your computer or compromise your data.

The security restrictions can be tricky to work with during development. When running a program locally, data can be read directly from the disk, though it must be placed relative to the user's "working directory," generally the location of the application. When running online, data must comefrom a location on the same server. It might be bundled with the codeitself (in a JAR archive, discussedlater; or from another URL on the sameserver). For a local file, Java's

`FileInputStream` classcan be used. If the file is bundled in a JAR archive, the `getResource()` function is used. For a file on the server, `URL.openStream()` might be employed. During the journey from development to deployment, it may be necessary to useall three of thesefunctions.

With Processing, eachof these scenarios (and some others) is handled transparently by the file API functions. By placing resourcesin the data folder, Processingpackages the files asnecessaryfor online and offline use.

File handling functions include `loadStrings()`, which readsa text file into an array of String objects, and `loadImage()` which reads an image into a PImage object, the container for image data in Processing.

```
// Examples of loading a text file and a JPEG image
// from the data folder of a sketch.
String[] lines = loadStrings("something.txt");
PImage image = loadImage("picture.jpg");
```

Theseexamples may be a bit easierto read if you know the programming conceptsof data types and classes. Each variable has to have a data type, such as `String` or `PImage`.

The `String[]` syntax means "an array of data of the class`String`." This array is created by the `loadStrings` function and is given the name lines; it will presumably be used later in the program under this name. The reason `loadStrings` createsan array is that it splits the something.txt file into its individual lines. The following function creates a single variable of class`PImage`, with the name image.

To add a file to the data folder of a Processingsketch, usethe Sketch→ Add File menu option, or drag the file into the editor window of the PDE. The data folder will be created if it doesnot exist already.

To view the contents of the sketch folder, usethe Sketch→ Show Sketch Folder menu option. This opensthe sketch window in your operating system'sfile browser.

Libraries add new features

A library is a collection of codein a specifiedformat that makesit easyto usewithin Processing. Libraries havebeenimportant to the growth of the project, becausethey let developers make new features accessibleto users without needing to make them part of the core ProcessingAPI.

Several core libraries come with Processing. These can be seen in the Libraries section of the online reference (also available from the Help menu from within the PDE.) These libraries can be seen at http://processing.org/reference/libraries/

One example is the PDF Export library. This library makes it possible to write PDF files directly from Processing. These vector graphics files can be scaled to any size and output at very high resolutions.

To use the PDF library in a project, choose Sketch → Import Library → pdf. This will add the following line to the top of the sketch:

```
import processing.pdf.*;
```

Java programmers will recognize the `import` command. In Processing, this line is also used to determine what code is packaged with a sketch when it is exported as an applet or application.

Now that the PDF library is imported, you may use it to create a file. For instance, the following line of code creates a new PDF file named lines.pdf that you can draw to.

```
beginRecord(PDF, "lines.pdf");
```

Each drawing function such as `line()` and `ellipse()` will now draw to the screen as well as to the PDF.

Other libraries provide features such as reading images from a camera, sending and receiving MIDI and OSC commands, sophisticated 3D camera control, and access to MySQL databases.

Sketching and scripting

Processing sketches are made up of one or more tabs, with each tab representing a piece of code. The environment is designed around projects that are a few pages of code, and often three to five tabs in total. This covers a significant number of projects developed to test and prototype ideas, often before embedding them into a larger project or building a more robust application for broader deployment.

The idea of sketching is identical to that of scripting, except that you're not working in an interpreted scripting language, but rather gaining the performance benefit of compiling to Java class files. Of course, strictly speaking, Java is itself an interpreted language, but its bytecode compilation brings it much closer to the "metal" than languages such as JavaScript, Python, or Ruby.

Processing was never intended as the ultimate language for programming visuals; instead, we set out to make something that was:

- A sketchbook for our own work, simplifying the majority of tasks that we undertake.
- A programming environment suitable for teaching programming to a non-traditional audience.
- A stepping stone from scripting languages to more complicated or difficult languages such as full-blown Java or C++.

At the intersection of these points is a tradeoff between speed and simplicity of use. If

we didn't care about speed, it might make sense to use Python, Ruby, or many other scripting languages. This is especially true for the education side. If we didn't care about making a transition to more advanced languages, we'd probably avoid a C++ or Java-style syntax. But Java makes a nice starting point for a sketching language because it's far more forgiving than C/C++ and also allows users to export sketches for distribution via the web.

Processing assembles our experience in building software of this kind (sketches of interactive works or data-driven visualization) and simplifies the parts that we felt should be easier, such as getting started quickly, and insulating new users from issues like those associated with setting up Java.

Don't start by trying to build a cathedral

If you're already familiar with programming, it's important to understand how Processing differs from other development environments and languages. The Processing project encourages a style of work that builds code quickly, understanding that either the code will be used as a quick sketch, or ideas are being tested before developing a final project. This could be misconstrued as software engineering heresy. Perhaps we're not far from "hacking," but this is more appropriate for the roles in which Processing is used. Why force students or casual programmers to learn about graphics contexts, threading, and event handling functions before they can show something on the screen that interacts with the mouse? The same goes for advanced developers: why should they always need to start with the same two pages of code whenever they begin a project?

In another scenario, the ability to try things out quickly is a far higher priority than sophisticated code structure. Usually you don't know what the outcome will be, so you might build something one week to try an initial hypothesis, and build something new the next based on what was learned in the first week. To this end, remember the following considerations as you begin writing code with Processing:

- Be careful about creating unnecessary structures in your code. As you learn about encapsulating your code into classes, it's tempting to make ever-smaller classes, because data can always be distilled further. Do you need classes at the level of molecules, atoms, or quarks? Just because atoms go smaller doesn't mean that we need to work at a lower level of abstraction. If a class is half a page, does it make sense to have six additional subclasses that are each half a page long? Could the same thing be accomplished with a single class that is a page and a half in total?
- Consider the scale of the project. It's not always necessary to build enterprise-level software on the first day. Explore first: figure out the minimum code necessary to help answer your questions and satisfy your curiosity.

The argument is not to avoid continually rewriting, but rather to delay engineering work until it's appropriate. The threshold for where to begin engineering a piece of software is much later than for traditional programming projects because there is a kind of art to the early process of quick iteration.

Of course, once things are working, avoid the urge to rewrite for its own sake. A rewrite should be used when addressing a completely different problem. If you've managed to hit the nail on the head, you should refactor to clean up function names and class interactions. But a full rewrite of already finished code is almost always a

bad idea, no matter how "ugly" it may seem.