PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

CENTRO TÉCNICO CIENTÍFICO – CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Ciência da Computação

Protótipo de sistema procedimental para geração e compartilhamento de características de personagens 2D

Raquel Olhovetchi Ferreira da Silva

Orientador: Prof. Augusto Cesar Espíndola Baffa

Rio de Janeiro

Julho de 2025

Raquel Olhovetchi Ferreira da Silva

Protótipo de sistema procedimental para geração e compartilhamento de características de personagens 2D

Relatório de Projeto Final,apresentado ao programa de **Ciência da Computação** da PUC-Rio como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Augusto Cesar Espíndola Baffa

Rio de Janeiro

Julho de 2025

Agradecimentos

Ao meu namorado Nagib que me ajudou em todas as etapas desse projeto, apontando onde poderia melhorar e estando sempre do meu lado nos melhores e piores momentos.

Aos meus amigos por escutarem madrugadas à fio minhas ideias mirabolantes e reclamações de códigos que não rodavam.

À Bárbara, cuja presença foi indispensável para meu crescimento pessoal e acadêmico.

À minha querida mãe que sempre me apoiou e confiou em mim, me dando forças para prosseguir com meus estudos.

Muito obrigada por tudo.

Resumo

Olhovetchi, Raquel. Geração Procedimental de Personagens 2D com Algoritmos Genéticos. Rio de Janeiro, 2025. 54 páginas. Relatório Final de Projeto Final de Graduação – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho apresenta o desenvolvimento de um sistema de geração procedimental de personagens 2D, com foco na aplicação de algoritmos genéticos para simular hereditariedade e variações evolutivas em um ambiente visual interativo. O sistema permite criar personagens compostos por sprites modulares, com atributos visuais e funcionais herdáveis, e possibilita a escolha entre diferentes métodos de seleção parental (roleta, torneio e ranking) e de crossover (single-point, two-point e scattered). A ferramenta foi implementada na engine Unity, enquanto a análise dos resultados foi realizada em Python via Kaggle. Foram realizados testes com nove populações simultâneas, utilizando diferentes combinações de algoritmos e ambientes, totalizando 450 mil indivíduos gerados. Os dados registrados em log permitiram análises de desempenho, convergência e diversidade genética. Os resultados demonstram que a escolha do método de crossover impacta diretamente a taxa de convergência, e que o método de seleção por torneio tende a gerar melhores desempenhos médios. O sistema também oferece uma funcionalidade de reprodução visual dos indivíduos registrados em log, o que torna a análise mais acessível e didática. A proposta integra aprendizado, visualização e análise comparativa, contribuindo para o estudo de algoritmos genéticos em contextos aplicados à geração de conteúdo em jogos.

Palavras-chave: geração procedimental, algoritmos genéticos, Unity, personagens 2D, simulação visual

Abstract

Olhovetchi, Raquel. Procedural Generation of 2D Characters with Genetic Algorithms. Rio de Janeiro, 2025. 54 pages. Undergraduate Thesis – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

This work presents the development of a procedural 2D character generation system, focusing on the application of genetic algorithms to simulate heredity and evolutionary variation within an interactive visual environment. The system enables the creation of characters composed of modular sprites with inheritable visual and functional attributes. It supports the use of various parent selection methods (roulette, tournament, and ranking) and crossover strategies (single-point, two-point, and scattered). The tool was implemented in the Unity engine, with result analysis conducted using Python on the Kaggle platform. Nine simultaneous populations were tested using different algorithm and environment combinations, totaling 450,000 individuals. Logged data enabled performance, convergence, and genetic diversity evaluations. Results show that the crossover method significantly affects convergence speed, and tournament selection tends to yield higher average fitness. The system also includes a visual playback feature to reconstruct characters from logs, making analysis more intuitive and didactic. This proposal integrates learning, visualization, and comparative analysis, contributing to the study of genetic algorithms in game-related procedural content generation.

Keywords: procedural generation, genetic algorithms, Unity, 2D characters, visual simulation

Sumário

1. Introdução

1.1. Motivação e definição do problema

A criação em massa de personagens e assets com características diferentes pode ser uma tarefa exaustiva, e quando feita de forma inconsistente, costuma ficar bastante perceptível no resultado final. Por isso, é fundamental estabelecer processos bem definidos para garantir que o produto atenda às expectativas de qualidade e imersão. Com o crescimento da indústria de jogos e a elevação da exigência por qualidade, a geração procedimental se consolidou como solução viável para reduzir custos e escopo (Shaker, Togelius e Nelson, 2016).

Esse método utiliza algoritmos para gerar componentes de forma dinâmica e semi-aleatória, sendo amplamente empregado na criação de mapas, plataformas e, em alguns casos, de personagens. Jogos de mundo aberto ou *dungeon crawler* frequentemente se beneficiam dessa abordagem — como acontece em *No Man's Sky*, que define planetas e fauna de modo procedural (Tait e Nelson, 2022), e em *Spore*, que cria criaturas com formas e comportamentos distintos (Kushner, 2008).

Nesse contexto, algoritmos genéticos surgem como alternativa interessante para controlar a variabilidade e coerência do conteúdo gerado. Ajustes como o método de seleção de pais, taxas de mutação e estratégias de cruzamento formam um conjunto de parâmetros que pode ser otimizado de acordo com os objetivos do projeto.

Algumas técnicas priorizam a diversidade entre os indivíduos, enquanto outras buscam soluções mais rapidamente otimizadas. Rawat et al. (2022) analisam métodos como seleção por roleta, torneio e ranking, mostrando como cada um se comporta em diferentes cenários. A seleção proporcional (*roulette wheel*) é eficiente para problemas matemáticos simples (Hermawanto, 2013), mas pode ser lenta em situações mais complexas (Goldberg e Deb, 1991).

A seleção por torneio foca na otimização rápida e costuma mostrar bom desempenho em problemas desafiadores — como demonstrado por Hu et al. (2005) no caixeiro-viajante, e por Champlin (2018) no dilema do prisioneiro. Já a seleção por ranking busca um equilíbrio entre diversidade e performance, sendo recomendada por Pandey (2016) para cenários que exigem variedade nas soluções.

Esses trabalhos comprovam que a seleção de técnica deve levar em conta os objetivos do sistema, como diversidade visual, rapidez de convergência ou equilíbrio entre ambos.

De forma geral, algoritmos genéticos se baseiam em um processo de evolução por gerações, onde cada indivíduo (solução candidata) passa por avaliação, seleção, cruzamento e mutação. Os mais adaptados sobrevivem ou se reproduzem, enquanto pequenas mutações garantem a diversidade e evitam que o sistema trave em soluções subótimas.

A adoção de algoritmos genéticos neste projeto se justifica por sua capacidade de explorar combinações visuais e funcionais dos personagens de modo controlado e eficiente. Em vez de buscar um único “personagem ideal”, a proposta é gerar múltiplos resultados e avaliar como diferentes estratégias afetam a aparência e o desempenho.

Para dar suporte a isso, o uso de *Scriptable Objects* no Unity foi essencial. Esses objetos funcionam como containers de dados reutilizáveis, permitindo definir atributos, sprites e configurações no editor e compartilhá-los entre prefabs e scripts sem duplicação de dados — além de facilitar ajustes em tempo real durante o desenvolvimento, sem necessidade de recompilação (Unity Manual, versão 6.1; GameDev Beginner, 2022).

O destaque visual do sistema permite acompanhar de forma intuitiva o impacto dos operadores genéticos. Assim, o projeto funciona tanto como ferramenta de pesquisa quanto como plataforma interativa para estudar algoritmos evolutivos, inteligência artificial, design procedural e desenvolvimento de jogos.

1.2. Sobre o ambiente computacional

Como a proposta do projeto testar o algoritmo em um protótipo de jogo, optei por utilizar a engine Unity devido à minha familiaridade com a plataforma.Assim, diversos recursos nativos foram utilizados para facilitar o desenvolvimento, como a implementação de recursos gráficos, testes e criação de *Scriptable Objects.*

Além do Unity, foi utilizada a plataforma Kaggle para criação de gráficos e análise de resultados.

1.3. Sobre a adequação do trabalho como projeto final

Este projeto permite a aplicação de diversos conceitos aprendidos ao longo da minha formação, com ênfase na área de inteligência artificial e geração procedimental. A implementação dos algoritmos genéticos envolve o uso de técnicas de seleção, mutação e crossover, demonstrando uma aplicação prática dos fundamentos teóricos aprendidos principalmente na disciplina de Inteligência Artificial.

O aspecto teórico do projeto inclui uma análise comparativa de diferentes métodos de criação, avaliando seus resultados em cenários práticos. Essas análises contam também com a presença de gráficos e visualizações detalhadas que ilustram o desempenho e as características dos métodos testados. Isso não apenas proporciona um entendimento mais profundo das técnicas existentes, mas também facilita a identificação de pontos fortes e limitações de cada abordagem.

Essa abordagem visual não apenas auxilia no aprendizado e na validação dos métodos utilizados, mas também torna o projeto uma ferramenta didática, servindo como uma base para futuras investigações ou aprimoramentos.

2. Situação atual

Durante a pesquisa sobre geração procedimental em jogos, foi observado que a maioria dos estudos se concentra na criação de mapas ou ambientes tridimensionais, enquanto a geração de personagens — especialmente em 2D — ainda é pouco explorada na literatura acadêmica. Essa ausência, destacada por Gaisbauer e Hlavacs (2017), aponta a predominância de pesquisas voltadas para terrenos, edifícios e vegetação, deixando de lado a criação automática de personagens. Na prática, essa lacuna se reflete no uso contínuo de métodos manuais em muitos projetos, sobretudo em jogos com estética bidimensional.

Mesmo entre os trabalhos que abordam personagens, os principais desafios identificados giram em torno da coerência visual, escalabilidade e diversidade do conteúdo gerado — fatores importantes para a manutenção da qualidade e a consistência em jogos. Por exemplo, Shan (2021) propõe um sistema de geração de personagens 3D segmentado por partes do corpo, com princípios hereditários. O autor destaca as dificuldades em manter a identidade visual dos personagens dentro de um universo estilizado e coerente. Situações semelhantes podem ser observadas por Tait e Nelson (2022), ao analisarem os desafios enfrentados pela equipe de *No Man's Sky* na criação de ambientes escaláveis que ainda transmitissem identidade e unidade visual. Já Kushner (2008), ao examinar o desenvolvimento de *Spore*, relata os obstáculos técnicos e criativos para gerar criaturas adaptáveis que respeitem tanto regras biológicas fictícias quanto a estética do jogo. Bach e Madsen (2007) também tratam da geração automática de personagens e da importância de garantir controle artístico no processo para que os resultados se integrem bem ao design geral.

Nesse contexto, estudar a geração de personagens 2D se mostrou uma oportunidade valiosa para preencher essa lacuna acadêmica, ao mesmo tempo em que se explora um formato visual mais acessível e popular em jogos independentes. Ao contrário da modelagem volumétrica e manipulação de malhas usada em 3D, o design 2D exige domínio de sprites, animações quadro a quadro e paletas de cor limitadas — elementos centrais em estilos como pixel art ou minimalista. Isso requer soluções próprias, e não apenas a adaptação direta de técnicas tridimensionais.

Com base nisso, este projeto propõe o desenvolvimento de um sistema de geração procedimental de personagens 2D, inspirado em conceitos

apresentados por Shan (2021), como a segmentação modular e a herança de atributos. Esses conceitos foram adaptados ao contexto bidimensional, permitindo que os personagens gerados visualizem de forma clara suas características herdadas, respeitando uma composição gráfica coerente.

A construção do algoritmo genético também foi guiada por estudos como o de Rawat et al. (2022), que comparam diferentes métodos de seleção — roleta, torneio e ranking — e analisam como essas escolhas impactam a diversidade e a performance das gerações. Tais fundamentos foram implementados para testar a eficácia de cada método no equilíbrio entre variedade visual e otimização evolutiva dos personagens.

Outro ponto central do sistema envolve os operadores de cruzamento genético. Trabalhos como os de Ambuehl (2017) e Park e Park (2021) mostram que variações como single-point, two-point e scattered crossover influenciam diretamente a convergência dos algoritmos e a manutenção da diversidade da população. Essas estratégias foram testadas no projeto, tanto em relação ao impacto visual quanto ao comportamento estatístico das gerações.

Dessa forma, o sistema desenvolvido neste trabalho foca na criação de personagens 2D utilizando algoritmos genéticos para simular visualmente princípios de hereditariedade. A proposta une visualização interativa, variação evolutiva e adaptabilidade gráfica, oferecendo uma ferramenta experimental que vai além da geração automatizada, permitindo a análise e manipulação de fatores evolutivos em um ambiente livre. Ao adaptar conceitos da geração procedural 3D ao universo 2D, este projeto contribui tanto para o desenvolvimento de jogos quanto para o estudo aplicado de algoritmos evolutivos em contextos visuais e estilizados.

3. Objetivos do trabalho

Esse projeto propõe o desenvolvimento de uma ferramenta de criação procedimental que simula um sistema de hereditariedade entre os personagens e mantêm o fator aleatório característico desse tipo de algoritmo. Essa

hereditariedade não se limita apenas às características físicas dos personagens, mas também se estende aos atributos como cor e altura que exploram essa mecânica como inspiração, como "*Clusterduck*"[1]. O propósito da ferramenta será estudar com auxílio visual o processo de geração genética, com variações na seleção parental e nos métodos de crossover.

O objetivo é implementar e comparar diferentes métodos de seleção (roleta, torneio e ranking), bem como aplicar diferentes estratégias de crossover (single-point, two-point e scattered) e gerar análises baseadas em resultados.

Dessa forma, de maneira dinâmica, poderá ser explorada a variação de aleatoriedade e suas implicações no resultado final.

Uma vez criada a ferramenta, o plano é integrá-la a um protótipo de jogo para avaliar não apenas sua utilidade teórica, mas também sua aplicação prática, com a inclusão de diversos métodos de criação que poderão ser manualmente escolhidos pelo usuário a fim de comparar os resultados finais de geração.

Ao final de cada processo de criação,será criado um LOG com todas as características dos personagens gerados, que poderá ser utilizado em ferramentas de análise de gráficos para comparar resultados.

Dessa forma, baseando-se em projetos de criação de personagem procedimental já existentes, o objetivo deste projeto é desenvolver um protótipo de sistema que permita estudar as técnicas de algoritmos genéticos e suas implementações práticas, enquanto adiciona um fator visual à criação de personagens, em um formato de jogo, garantindo simultaneamente um gerador procedimental personalizado com características genéticas herdadas e um sistema de estudo intuitivo para os usuários.

[1] 2022, PikPok

4. Atividades Realizadas

4.1. Estudos conceituais e de tecnologia

Foram separados os estudos em duas vertentes, a primeira sendo a criação de personagens em si, adaptando os conceitos de geração procedimental para a criação de bonecos que pudessem ser representados visualmente.O segundo estudo baseou-se nas técnicas de geração procedimental em si; uma vez que a criação prototipada dos bonecos estava finalizada, foquei em como poderia utilizá-la em jogos ou simuladores, e iniciei uma pesquisa sobre diferentes métodos de escolha de pais, crossover, mutação e demais aspectos que englobam a geração procedimental.

Baseando-me nas pesquisas do artigo “A Comparative Review Between Various Selection Techniques In Genetic Algorithm For Finding Optimal Solutions” pude observar os prós e contras de diversos métodos de seleção de pais e aplicá-los no código de maneira a confirmar a veracidade dos resultados vistos nos demais artigos.

Com base nos resultados obtidos por Park e Park (2021), que compararam empiricamente métodos de seleção e crossover em algoritmos genéticos, pode-se observar que a escolha do método de crossover tem impacto mais significativo sobre a taxa de convergência do que a escolha do método de seleção. Tais conclusões foram utilizadas como base para a priorização da análise dos métodos de crossover no presente trabalho, buscando avaliar sua influência sobre o tempo de convergência e a diversidade dos personagens gerados.

4.2. Criação de Assets

Para a criação dos personagens no projeto, foi utilizado o conceito de bonecos de papel, que consiste na separação dos diferentes segmentos de um personagem (como cabeça, braços, pernas, tronco, olhos, boca, etc.), permitindo que essas partes sejam combinadas de forma modular para gerar uma grande diversidade de personagens.

Esse método foi inspirado em jogos como Clusterduck e plataformas como Picrew, onde são selecionadas e combinadas diferentes partes do corpo e características visuais para criar um personagem único. No projeto, esse processo foi automatizado para ser feito de forma procedimental, utilizando um banco de dados de imagens autoral e atributos para gerar personagens com variações de cor, tamanho e formato.Esses assets foram desenhados utilizando a ferramenta Clip Studio, e cada parte do corpo foi projetada separadamente, com variações de estilos e atributos. Além disso, foi incluída a possibilidade de alteração de cores e formas para aumentar ainda mais a diversidade dos personagens gerados.

Alguns dos assets criados podem ser observados na imagem 1, com exemplos de cabeças desenhadas.

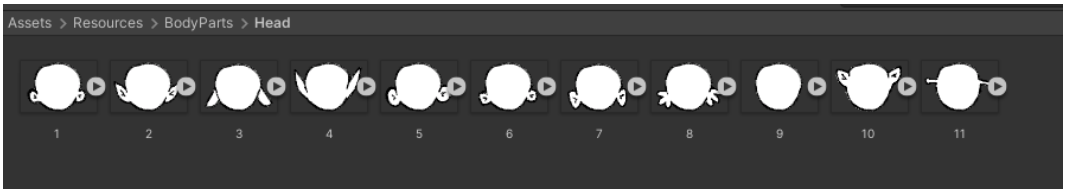

figura 1. exemplo de assets de cabeça utilizados no projeto

4.3. Desenvolvimento da Ferramenta de Geração Procedimental

A primeira etapa fundamental no desenvolvimento do projeto foi a criação da ferramenta capaz de gerar os personagens de forma procedimental. Utilizando as imagens e atributos criados, foi implementada a funcionalidade que permite a criação de personagens com diferentes partes do corpo separadas (cabeça, braços, pernas, tronco, olhos, boca, etc). Essas partes foram combinadas de forma procedimental para gerar um novo personagem, que foi armazenado em um *Scriptable Object* no Unity.

Além da variação dos sprites de cada segmento, a ferramenta permite mudanças de cor e tamanho para gerar mais diversidade entre os personagens. O algoritmo também foi projetado para incluir atributos não visuais, como vida,

força, agilidade e velocidade, com valores variáveis dentro de limites arbitrários, adicionando uma camada extra de complexidade aos personagens gerados.

4.4. Incorporação de Hereditariedade

A etapa seguinte focou na incorporação de hereditariedade, onde as características físicas e de atributos dos personagens "filhos" deveriam ser similares aos dos pais, com a introdução de um nível controlado de aleatoriedade para simular mutações genéticas. O algoritmo não seria completamente determinístico, garantindo que os resultados de cada simulação fossem variados, mas com uma consistência nas características herdadas.

O comportamento da hereditariedade foi testado para verificar se, para as mesmas entradas, os resultados eram reproduzíveis, antes de introduzir o fator aleatório que simula as mutações.

4.5. Polimento do algoritmo

A partir da aplicação de técnicas de algoritmos genéticos, foi necessário polir o comportamento do sistema para garantir que os personagens gerados estivessem alinhados com as métricas e objetivos estabelecidos. Segundo Bach e Madsen (2015), "os algoritmos genéticos dependem de uma função de aptidão e não podem ser completamente automatizados, já que o artista precisa de controle". Nesse contexto, a função de aptidão foi ajustada para permitir ao usuário controlar as soluções ideais para cada execução do algoritmo.

O algoritmo foi aprimorado para analisar se os atributos dos personagens estavam de acordo com o esperado, se a montagem dos personagens era visualmente agradável e se o sistema de hereditariedade funcionava corretamente. Além disso, o fator aleatório foi introduzido gradualmente, para avaliar se os resultados ainda seguiam as métricas de qualidade preestabelecidas.

4.6. Testes e Protótipos para aprendizado e demonstração

Como protótipo para o projeto, foi desenvolvida uma cena no Unity que contém os personagens criados procedimentalmente, com diversas possibilidades de alteração do ambiente, envolvendo fatores externos como temperatura, cor de chão, probabilidade de mutação, tamanho da população. Uma vez que também era o interesse demonstrar visualmente os estudos feitos, foi optado por não escolher apenas um método de seleção de pais,crossover e mutação, deixando esses atributos à escolha do usuário para que possa fazer sua própria análise. Dessa forma, modificadores estruturais como a escolha de método de seleção de pais (variando entre torneio, roleta e rank) também são mutáveis.

Após a criação, foi determinado que seria gerado um LOG escrito com os dados de todos os membros da população, para que pudesse ser posteriormente analisado por um script em Python criado na plataforma Kaggle, verificando fatores como fitness, variações de cor, características de temperatura e tamanho.Dessa forma, podendo-se analisar os resultados referentes a cada variação e cruzando com as informações obtidas em artigos de análise.

Além disso, foi desenvolvido um script complementar dentro do Unity responsável por importar os dados de log gerados em formato CSV e recriar visualmente os personagens descritos nesses arquivos. Essa funcionalidade permite, além da análise estatística dos dados, a visualização completa de qualquer indivíduo registrado, facilitando a inspeção visual de características genéticas herdadas, mutações e padrões de convergência. A ferramenta funciona como um reprodutor de gerações passadas, sendo útil para testes comparativos e validação dos resultados obtidos.

A cena criada no Unity para manipulação das populações pode ser observada na imagem 2.

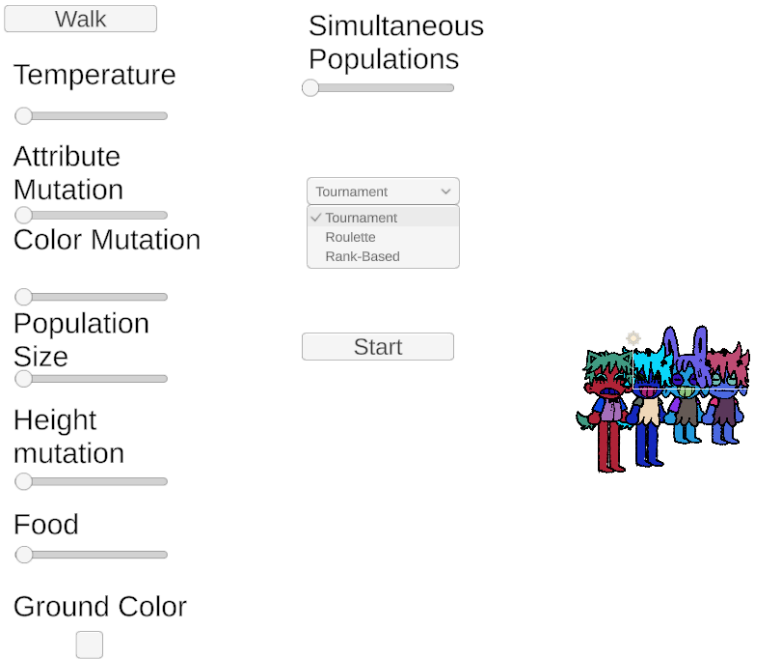

figura 2. cena criada no Unity para testes

5. Projeto e especificação do Sistema

A proposta do projeto surgiu da ideia de criar um sistema de geração de personagens 2D de forma prática, utilizando conceitos de algorítmos genéticos para simular herança, seleção e variedade de características. Além de servir como forma de estudo e experimentação de diferentes estratégias evolutivas, também apresenta o contexto visual e interativo que contribui para a melhor compreensão dos resultados.
O sistema foi desenvolvido na engine Unity, utilizando recursos como Scriptable Objects para facilitar a modularização do projeto.

A população inicial é gerada de forma aleatória, e a partir da mesma,as demais são geradas por meio da combinação genética de dois progenitores, seguindo então o fluxo típico de um algoritmo genético-seleção dos pais, cruzamento (crossover) e mutação. Os resultados podem ser observados visualmente através de personagens que populam o cenário. Além disso, o sistema permite o ajuste manual dos parâmetros como taxa de mutação, temperatura, cor do ambiente, bem

como a seleção manual das formas de seleção e métodos de cruzamento, garantindo assim uma análise prática de como as configurações afetam o resultado.

5.1. Descrição das classes

Esta seção descreve as principais classes implementadas no simulador, abordando seus atributos, responsabilidades e métodos. A figura 3 apresenta o diagrama de classes, que inclui as classes PopulationGeneration, GenerateCharacter,AttributesList, CharacterBody, Stats e Character

Além disso possui referência à classe CrossoverStrategies, e GenerateCharacters

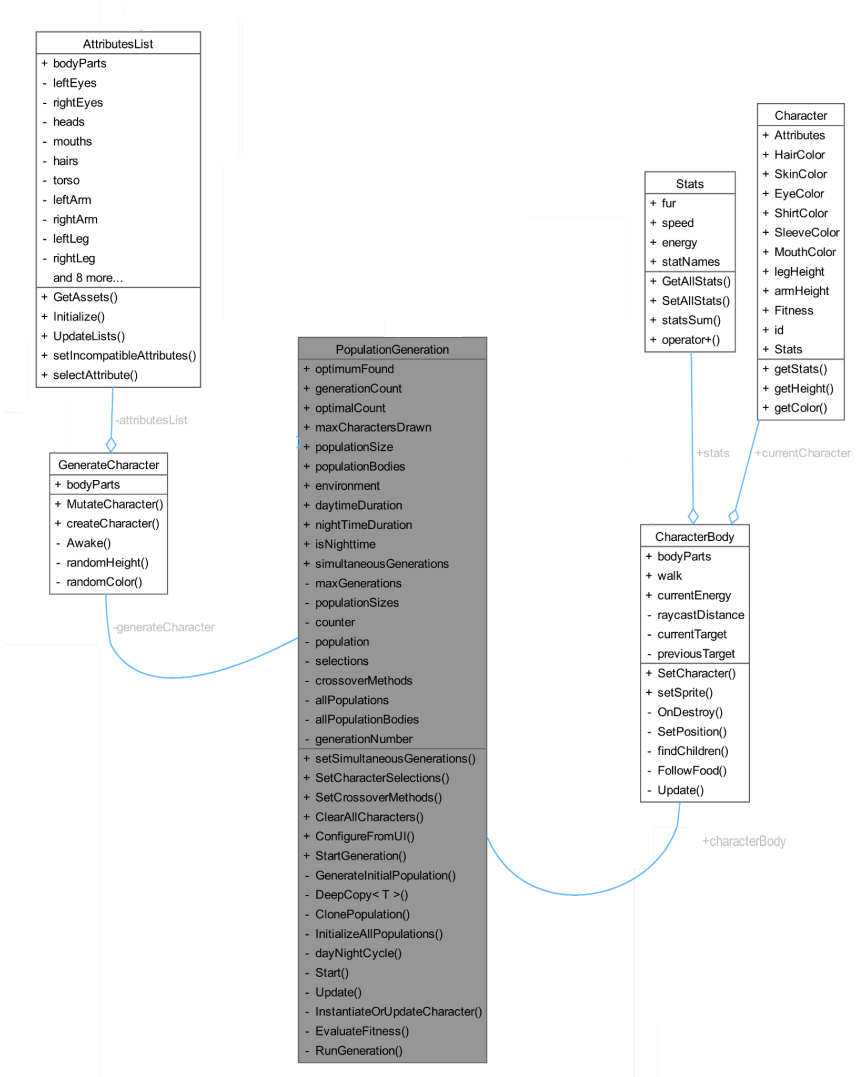

figura 3. Diagrama de colaboração de classes do sistema

5.1.1 .Classe Stats

A classe Stats representa os atributos quantitativos dos personagens: pelagem (fur) e vision(visão). Esses valores são utilizados na avaliação de fitness e influenciam o desempenho dos indivíduos no ambiente.
Atributos principais:

- **fur**: Adaptação térmica.
- **Vision**: visão.

5.1.2 .Classe Character

A classe Character encapsula todas as informações genéticas e visuais de um personagem gerado, incluindo atributos físicos, cores e valor de fitness. Ela serve como modelo lógico que será posteriormente instanciado visualmente pela *CharacterBody*.
Além dos atributos quantitativos (Stats), a classe inclui características como cor de pele, cabelo, olhos e roupas, além de um identificador único (id) e um valor de aptidão (Fitness) calculado a cada geração.
Atributos principais:

- **id:** Identificador único do personagem.
- **Fitness**: Valor de aptidão avaliado durante a simulação.
- **Stats**: Atributos numéricos (pelagem, visão, velocidade).
- **SkinColor, HairColor, EyeColor, ShirtColor, SleeveColor, MouthColor:** Representações visuais do personagem.
- **LimbHeights**: Tamanhos dos membros, usados para variação visual.
  Métodos implementados:
  - **GetHeight():** Retorna a altura (escala vertical) de um membro específico do corpo
  - **GetStats():**Retorna o valor de um *stat* específico. Acessa os dados da instância de *Stats*.
  - **GetColor():**Retorna a cor correspondente ao nome especificado

5.1.3 .Classe CharacterBody

A classe *CharacterBody* é responsável por representar visualmente um personagem no ambiente Unity, instanciando os dados

definidos na classe Character. Ela conecta atributos genéticos com componentes gráficos (sprites) e ajusta cores, tamanhos e posicionamentos dos membros corporais.
Essa classe torna os personagens visíveis na simulação, aplicando os dados de cor e altura em elementos como cabeça, braços, pernas e corpo.

Métodos implementados:

- **SetCharacter():**Aplica os dados genéticos ao corpo visual do personagem, chamando internamente *SetHeights* e *SetSprite* para configurar escala e sprites.
- **SetSprite()**:Define o sprite de cada parte do corpo usando *SpriteRenderer.sprite* do Unity.

A figura 4 demonstra o Diagrama da classe CharacterBody.

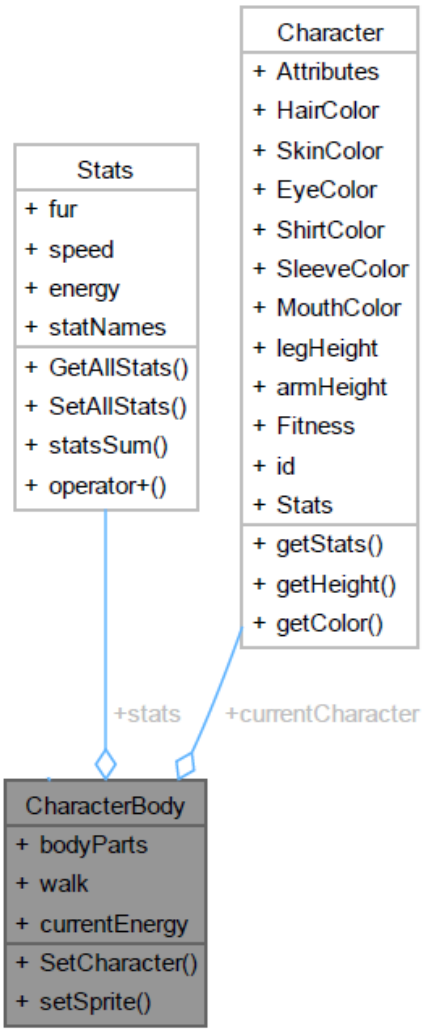

figura 4. Diagrama da classe CharacterBody

5.1.4 .Classe CrossoverStrategies

A classe *CrossoverStrategies* implementa os diferentes métodos de cruzamento genético utilizados na geração de novos personagens, a partir de

dois indivíduos pais. Ela define como os atributos dos pais são combinados para formar os descendentes, influenciando diretamente a diversidade genética da população.

Trata-se de uma classe utilitária, composta exclusivamente por métodos estáticos.

O método principal se trata do *Crossover,*que realiza o cruzamento entre dois personagens, gerando um número específico de descendentes. O parâmetro method define o tipo de crossover aplicado:

- 0: Single-point crossover
- 1: Two-point crossover
- 2: Scattered crossover

5.1.5 .Classe GenerateCharacter

A classe *GenerateCharacter* é responsável pela criação e mutação de personagens. Ela atua gerando combinações aleatórias ou modificadas de atributos genéticos a partir de regras definidas para cruzamento, mutação e variabilidade.

Métodos principais implementados:

- *CreateCharacter()*:Gera um novo personagem com atributos aleatórios, cores variadas e proporções corporais randômicas. É usado na criação da população inicial.
- *MutateCharacter():* Aplica mutações ao personagem .

5.1.6 .Classe PopulationGeneration

A classe *PopulationGeneration* é o núcleo do sistema evolutivo. Ela coordena todo o ciclo de vida das populações de personagens: criação inicial, seleção, cruzamento, mutação, avaliação e visualização. Também gerencia múltiplas populações em paralelo, permitindo comparar diferentes combinações de estratégias evolutivas.

É implementada como um MonoBehaviour no Unity, integrando o sistema ao ambiente visual e ao tempo de simulação

Métodos principais implementados:

- *RunGeneration*():Executa uma geração de cruzamento e avaliação para a população
- *EvaluateFitness*(): Calcula a aptidão de um personagem com base na semelhança de cor com o ambiente e na diferença de temperatura.

5.2. Criação de personagens

Para a criação dos personagens, foram definidos atributos que os constituem, os quais são classificados em atributos visuais e não visuais, como vida, velocidade, altura e características físicas, tais como olhos, boca e nariz. Cada personagem pode ser gerado de duas maneiras: de forma aleatória ou mediante a combinação genética de dois progenitores. A combinação a partir de dois progenitores envolve a aplicação de métodos do algoritmo genético, incluindo seleção dos progenitores, crossover e mutação.

Os personagens gerados pelo sistema possuem atributos visuais e não-visuais que influenciam tanto na aparência quanto no comportamento durante a simulação. A estrutura dos personagens é definida pela classe Character, onde os atributos são armazenados e manipulados pelo algoritmo genético.

Na figura 5 são apresentados os principais atributos de cada personagem:

Tipo	Atributo	Descrição
Visual	HairColor	Cor do cabelo e cauda(se existente)
Visual	SkinColor	Cor da pele (tronco, braços, pernas, cabeça, mãos e pés)
Visual	EyeColor	Cor dos olhos
Visual	ShirtColor	Cor da camiseta
Visual	SleeveColor	Cor das mangas
Visual	MouthColor	Cor da boca
Visual	ArmHeight	Altura dos braços
Visual	LegHeight	Altura das pernas
Stats	Speed	Velocidade de

		deslocamento do personagem
Stats	Energy	Quantidade de energia atual (gasta ao andar e recuperada ao comer)
Stats	Vision	Quantidade de visão em um ambiente iluminado
Stats	Fitness	Valor da aptidão, calculado com base em desempenho e fatores ambientais
Estrutural	Attributes[]	Lista de componentes visuais com sprites, posição e escalas individuais

figura 5. Principais atributos dos personagens

Além disso, cada personagem possui um conjunto de atributos herdados de seus "pais", com mutações aplicadas de forma aleatória controlada. A altura de pernas e braços, por exemplo, é usada na composição dos atributos speed e energy, conforme calculado pela função getStats().

A renderização e aplicação dos sprites são feitas dinamicamente, posicionando corretamente os membros e ajusta escalas conforme os valores herdados e mutados.

Para a criação dos atributos, optou-se pela utilização de *scriptable objects* para que facilitasse a modificação, retirada e adição de novos objetos.

A tabela contendo todos os atributos utilizados na base de dados e seus respectivos valores pode ser consultada no apêndice A

5.3. Os métodos de seleção

Na fase de seleção dos progenitores, foram implementados três métodos: o método de roleta, o método de ranking e o método de torneio.

Método de Roleta:

A seleção por roleta é baseada na ideia de que indivíduos mais aptos devem ter maior probabilidade de serem escolhidos para se reproduzir. O cálculo da probabilidade é proporcional ao valor de fitness de cada indivíduo. Assim, indivíduos com maior desempenho na simulação ocupam uma “fatia” maior no processo seletivo.Nesse método, cada indivíduo recebe uma chance de ser selecionado proporcional à sua aptidão. Indivíduos com maior aptidão possuem uma maior probabilidade de serem escolhidos para gerar a próxima geração.

Esse método é comparável a uma roleta, onde as probabilidades de cada indivíduo são determinadas de acordo com sua aptidão relativa.
Segundo Park e Park (2021), a probabilidade de um indivíduo ser selecionado é dada pela fórmula:

$$\rho(i) = f(i) / \sum_{j=1}^{n} f(j)$$

Onde:

- $f(i)$ é $o\ valor\ de\ fitness\ do\ indivíduo\ i$
- n é $o\ tamanho\ da\ população.$

Seu pseudocódigo é mostrado na figura 6.

```
seleção_roleta (P)
{
    totalFitness ← 0;

    para cada indivíduo i em P faça
    {
        totalFitness ← totalFitness + fitness(i);
    }

    para k de 1 até 2 faça
    {
        pick ← valor_aleatório(0, totalFitness);
        acumulador ← 0;

        para cada indivíduo i em P faça
        {
            acumulador ← acumulador + fitness(i);
            se (acumulador ≥ pick) então
            {
                adiciona i à lista selecionados;
                pare o loop interno;
            }
        }

}
```

figura 6. Pseudocódigo do método de seleção de roleta

Este método é sensível à escala do fitness, podendo sofrer com dominância quando há indivíduos com valores muito altos.

Método de Ranking: Para contornar os problemas de escalonamento da roleta, o método de ranking ordena os indivíduos da população com base em seus valores de fitness e atribui as chances de seleção a partir dessa ordenação. O fitness absoluto deixa de ser relevante, e o que importa é a posição relativa de cada indivíduo.

Esse tipo de seleção, como argumentam Goldberg e Deb (1991),proporciona uma pressão seletiva mais constante. Isso se mostrou útil neste trabalho, especialmente em populações com variabilidade de fitness acentuada, onde a roleta poderia favorecer exageradamente poucos indivíduos.

A aplicação desse método se mostrou eficiente para manter a diversidade, já que mesmo indivíduos medianamente aptos mantêm uma chance razoável de serem selecionados.

Ao invés de basear as chances de seleção diretamente na aptidão absoluta de um indivíduo, o método de ranking classifica todos os indivíduos em uma ordem, do mais apto ao menos apto. As probabilidades de seleção são então atribuídas com base na classificação do indivíduo. Esse método pode ser útil para evitar uma sobrecarga de seleção dos indivíduos extremamente aptos, que podem dominar a população.
Como pode ser visto em Park e Park (2021); A probabilidade de um indivíduo ser selecionado é dada pela fórmula:

$$\rho(i) = rank(i)/n * (n - 1)$$

Onde p é a probabilidade do indivíduo i e n é o número total de indivíduos competindo.

Seu pseudocódigo é mostrado na figura 7.

```
seleção_ranking (P)
{
    ordena P por fitness crescente → lista_ordenada;
    n ← tamanho de P;
    totalRank ← 0;

    para i de 1 até n faça
    {
        totalRank ← totalRank + i;
    }

    para k de 1 até 2 faça
    {
        pick ← valor_aleatório(0, totalRank);
        acumulador ← 0;

        para i de 0 até n - 1 faça
        {
            acumulador ← acumulador + (i + 1);
            se (acumulador ≥ pick) então
            {
                adiciona lista_ordenada[i] à lista selecionados;
                pare o loop interno;
            }
        }
    }
}
```

figura 7. Pseudocódigo do método de seleção de ranking

Essa abordagem evita a dominância dos indivíduos mais aptos, promovendo mais diversidade (TOGELIUS et al., 2010).

Método de Torneio:

A seleção por torneio consiste em realizar pequenos “duelos” entre subconjuntos da população. Em cada torneio, dois ou mais indivíduos são sorteados aleatoriamente, e o mais apto entre eles é escolhido como progenitor.

Esse método foi apontado por Gabriel (2015) como eficiente e simples, além de fácil paralelização. A principal vantagem é a flexibilidade ao ajustar o tamanho dos torneios, sendo possível controlar a pressão seletiva. Torneios maiores favorecem os indivíduos mais aptos, enquanto torneios menores aumentam a aleatoriedade, preservando a diversidade da população.

Além disso, como discutido por Abulail (2025), esse tipo de seleção se adapta bem a ambientes com mutações dinâmicas, fator existente nesse projeto, pois o controle do tamanho dos torneios pode ser ajustado ao longo do tempo, dependendo do comportamento da população.

Neste método, um grupo de indivíduos é selecionado aleatoriamente e compete entre si, onde o vencedor (o mais apto) é escolhido para gerar a próxima geração. Esse processo pode ser repetido várias vezes para garantir uma diversidade maior entre os indivíduos selecionados.

Neste trabalho, optou-se por utilizar k=3. Segundo Park e Park (2021), tamanhos de torneio pequenos, especialmente entre 2 e 4, promovem um bom equilíbrio entre pressão seletiva e diversidade genética. Com k=3, ainda existe uma probabilidade razoável de indivíduos medianos ou até menos aptos serem selecionados ocasionalmente, o que mantém a variação genética e reduz o risco de convergência prematura.

Além disso, Park e Park (2021) apontam que, embora torneios maiores acelerem a convergência ao favorecer fortemente os indivíduos mais aptos, essa estratégia pode levar a um esgotamento do espaço de busca e, consequentemente, à estagnação do processo evolutivo. Portanto, a escolha por k=3k visa proporcionar estabilidade e robustez ao processo de seleção.

Seu pseudocódigo é mostrado na figura 8.

```
seleção_torneio (P, selecionados)
{
    k ← 3;  // tamanho do torneio

    para s de 1 até 2 faça
    {
        torneio ← lista vazia;

        para j de 1 até k faça
        {
            sorteia aleatoriamente um indivíduo i de P
            adiciona i à lista torneio;
        }

        melhor ← torneio[0];

        para cada indivíduo c em torneio faça
        {
            se (fitness(c) > fitness(melhor)) então
            {
                melhor ← c;
            }
        }

        adiciona melhor à lista selecionados;
    }
}
```

figura 8. Pseudocódigo do método de seleção de torneio

É um dos métodos mais simples e eficientes, ele gera bons indivíduos, ainda mantendo aleatoriedade.Ele apresenta boa performance geral em cenários variados (Rawat et al., 2022; Gabriel, 2015)

5.4. Os métodos de crossover

A etapa de crossover é responsável por combinar os atributos genéticos de dois progenitores selecionados, gerando novos indivíduos que serão avaliados na próxima geração. Esse processo visa preservar boas características já encontradas enquanto introduz variações que favoreçam a busca por soluções mais adaptadas. A implementação de diferentes métodos de crossover permite observar empiricamente seus impactos sobre a diversidade genética e a eficiência da evolução.

Neste trabalho, foram implementadas três estratégias de crossover: **single point**, **two point** e **scattered**. A seguir, descrevem-se seus princípios de funcionamento e características principais.

Single Point Crossover (Crossover de um ponto):

Nesse método, um ponto de corte é escolhido aleatoriamente ao longo da cadeia genética dos pais. Os genes localizados antes desse ponto são herdados do primeiro pai, e os posteriores, do segundo. Trata-se de uma estratégia simples, amplamente utilizada em algoritmos genéticos, e que tende a preservar blocos de genes localizados no início da sequência.

Apesar de eficiente, esse modelo pode ter limitações quando características importantes estão dispersas em regiões distantes da cadeia genética (SOON et al., 2013).

Parent 1: 0 1 0 1 | 1 0 1 1 0

Parent 2: 0 0 1 1 | 1 0 0 1 0

Offspring 1: 0 1 0 1 | 1 0 0 1 0

Offspring 2: 0 0 1 1 | 1 0 1 1 0

Two Point Crossover (Crossover de dois pontos):

Neste método, dois pontos de corte são escolhidos aleatoriamente. O segmento intermediário da sequência genética é trocado entre os pais. Essa estratégia permite uma recombinação mais ampla das informações genéticas, preservando parte das estruturas originais enquanto introduz variação significativa nos descendentes.

Parent 1: 0 1 | 0 1 1 0 | 1 1 0

Parent 2: 0 0 | 1 1 1 0 | 0 1 0

Offspring 1: 0 1 | 1 1 1 0 | 1 1 0

Offspring 2: 0 0 | 0 1 1 0 | 0 1 0

Scattered Crossover (Crossover disperso):

No crossover disperso, não há pontos de corte fixos. Cada gene do descendente é escolhido aleatoriamente entre os genes correspondentes dos

pais. Isso resulta em uma recombinação altamente variada, que favorece a diversidade da população e reduz a chance de estagnação precoce.

Esse tipo de crossover é útil nos estágios iniciais da evolução, quando a diversidade genética é essencial (BYE et al., 2020).

Parent 1: *1 1 0 1 1 0 1 1 0*

Parent 2: *0 0 1 1 1 0 0 1 0*

Offspring 1: *1 0 0 1 1 0 1 1 0*

5.5. Os métodos de mutação

Além da seleção e do cruzamento, a mutação é uma etapa fundamental no algoritmo genético, responsável por inserir variações aleatórias nos indivíduos e manter a diversidade na população. Diferente dos métodos estáticos de crossover e seleção parental decididos antes das gerações, nesse projeto a mutação pode ser modificada a qualquer momento, garantindo que mudanças visuais nos personagens possam ser vistas ao vivo, conforme os valores do ambiente são alterados em tempo real.

Por isso, as taxas de mutação foram implementadas como variáveis que vão de 0% a 100%, e podem ser modificadas diretamente pela interface. Isso permite observar de forma prática como diferentes configurações ambientais afetam o visual e os atributos dos personagens gerados.

Foram implementadas três categorias de mutação diferentes:

- **Mutação de atributos** (como velocidade, visão e pelagem);
- **Mutação de características visuais** (como cor da pele, dos olhos, cabelo, camisa e boca);
- **Mutação de proporções corporais** (como altura dos braços e pernas).

Durante a criação de um novo personagem, cada gene tem uma chance de sofrer mutação com base nesses valores. Se a mutação acontecer, o valor é substituído por um novo, sorteado aleatoriamente dentro de um intervalo aceitável. Por exemplo, uma cor pode mudar completamente para outra gerada aleatoriamente, e a altura de uma perna pode variar dentro dos limites mínimos e máximos definidos.

5.6. Protótipo do jogo

O sistema desenvolvido consiste em uma simulação interativa de evolução populacional de personagens 2D, modelada por meio de algoritmos genéticos e projetada dentro da engine Unity.

Seguindo os conceitos dos algoritmos genéticos, a população inicial é gerada de forma aleatória.Cada personagem é composta por atributos genéticos diversos, como velocidade, visão do personagem, densidade de pelagem, altura dos membros e cores visuais (pele, cabelo, roupa, olhos e boca). A estrutura visual dos personagens é representada pela classe *CharacterBody*, que conecta os dados genéticos a sprites 2D. Esse corpo é composto por membros instanciados em diferentes *Transform*, que permitem o ajuste dinâmico de escala e posição dos elementos visuais (Unity Technologies, 2025).

O tamanho da população é ajustável via interface, sendo 2 o valor padrão inicial. A cada ciclo de geração, os indivíduos são submetidos ao processo de reprodução evolutiva, conforme o fluxo clássico de algoritmos genéticos: seleção de progenitores, cruzamento (crossover) e mutação. Para essa etapa, foram implementados diversos métodos de seleção parental — como roleta, torneio e ranking — cujos comportamentos foram inspirados em estudos como os de Goldberg e Deb (1991), Park e Park (2021) e Gabriel (2015).

Da mesma forma, a classe *CrossoverStrategies* permite a escolha entre múltiplas estratégias de recombinação genética: crossover de ponto único, de dois pontos e disperso. A diversidade visual e funcional dos descendentes é garantida pelas operações de mutação com valores ajustáveis diretamente pela

interface do protótipo, conforme os princípios defendidos por Abulail (2025) para evitar a convergência prematura.

Além disso, o projeto possui um sistema de simulação paralela de múltiplas populações. Por meio da classe *PopulationGeneration*, o usuário pode definir quantas populações deseja gerar simultaneamente (por padrão, duas), cada uma com parâmetros independentes de seleção, crossover e mutação. Esse modelo permite observar, em tempo real, os efeitos comparativos entre diferentes estratégias evolutivas, promovendo uma análise empírica rica e acessível.

Além disso, o sistema foi projetado para operar em ciclos de dia e noite, onde a etapa de seleção e cruzamento ocorre durante o "dia", e a pausa de avaliação é representada pela "noite". A duração de cada fase é controlada por parâmetros ajustáveis, permitindo diferentes cadências de evolução.

Outro componente fundamental do protótipo é a classe Environment, que simula fatores ambientais como a temperatura e a cor do solo. Esses elementos são usados como referência para calcular a aptidão (fitness) dos personagens através de uma fórmula baseada em adaptação térmica (diferença entre temperatura ambiente e atributo de pelagem) e camuflagem (distância entre a cor da pele do personagem e a cor do solo). A função de avaliação está contida dentro da classe *PopulationGeneration* e influencia diretamente a probabilidade de seleção dos indivíduos para reprodução, conforme as regras definidas pelos algoritmos genéticos.

Concomitantemente à execução da simulação, o sistema realiza o registro estruturado de todas as informações de cada indivíduo por geração, por meio de um processo de exportação em CSV.Esses logs são salvos separadamente para cada população, incluindo dados como identificador, atributos genéticos, fitness, características visuais e ambientais.

Esses arquivos de log são posteriormente analisados por meio de scripts desenvolvidos em Python, disponíveis na plataforma Kaggle, que permitem a geração de gráficos e estatísticas de evolução. Um exemplo desses gráficos pode ser visto na imagem 7, que mostra a evolução do fitness médio ao longo das gerações no conjunto de dados referente à quinta população. A visualização desses dados facilita a compreensão do comportamento das estratégias

evolutivas ao longo do tempo, evidenciando padrões de convergência, variações genéticas e estabilidade populacional.

A figura 9 apresenta a interface da simulação com duas populações em execução paralela, cada uma exibindo seus indivíduos lado a lado. Esse arranjo visual permite que o usuário compare, de maneira intuitiva, o impacto de diferentes métodos de seleção e recombinação genética no fenótipo dos personagens ao longo das gerações.

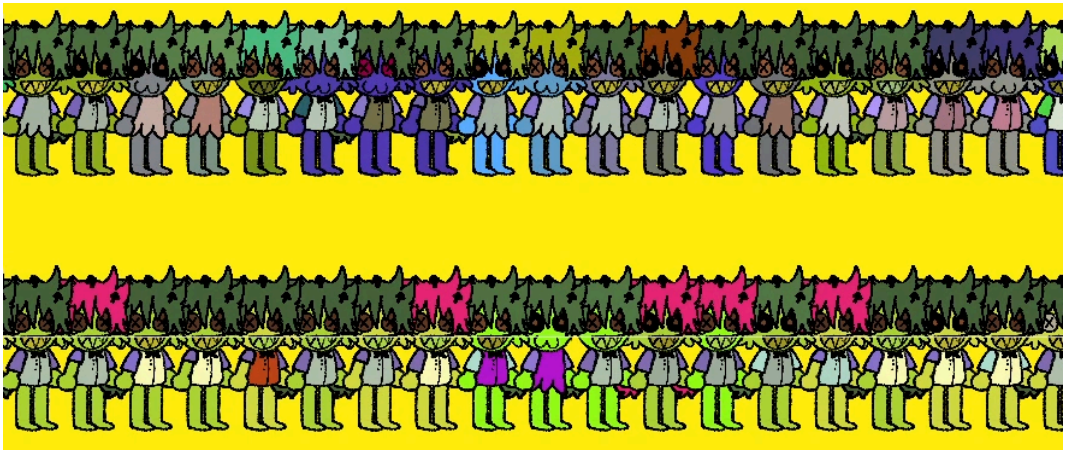

figura 9. interface da simulação com duas populações em execução paralela

A figura 10 demonstra um exemplo de gráfico criado para análise mais aprofundada dos resultados das gerações, focando no fitness médio por geração do dataset 5.

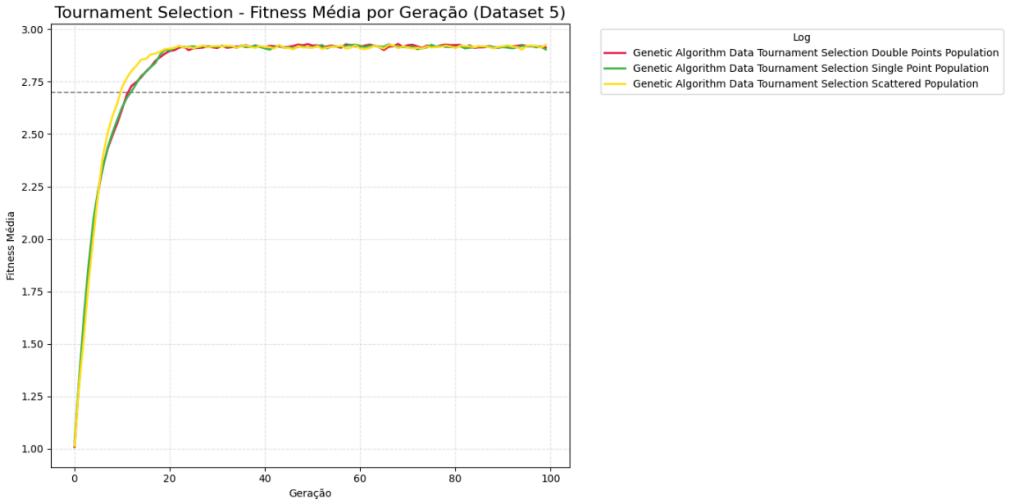

figura 10. exemplo de gráficos criados

6. Planejamento e Execução de Testes Funcionais

Os testes do sistema foram conduzidos de forma estruturada e imparcial para avaliar a eficácia e o desempenho dos algoritmos genéticos implementados.

Utilizando um script em Python na ferramenta Kaggle, colocando os resultados de cada simulação, foi possível analisar graficamente a evolução das populações simultaneamente, garantindo que estariam de acordo com os resultados esperados.

Cada componente foi testado de forma independente, com foco em diferentes etapas do processo:

1. Seleção de Pais: Os métodos de seleção (roleta, ranking e torneio) foram analisados individualmente para verificar se os pais escolhidos refletiam corretamente os critérios de aptidão. Logs detalhados foram utilizados para registrar os resultados e identificar possíveis discrepâncias.
Baseando-se no artigo “A Comparative Review Between Various Selection Techniques in Genetic Algorithm for Finding Optimal Solutions”, o resultado esperado era de que
2. Crossover: Os métodos de crossover (SinglePoint, TwoPoint e Scattered) foram avaliados separadamente. O objetivo foi verificar se os atributos combinados de dois progenitores eram coerentes com a lógica do método escolhido e se as variações esperadas estavam sendo geradas corretamente.De forma semelhante aos resultados reportados por Park e Park (2021), observou-se neste trabalho que o uso do crossover double point promove convergência significativamente mais rápida. Em contrapartida, o método scattered apresentou maior diversidade nas gerações iniciais, porém convergiu de forma mais lenta. Isso valida a hipótese de que a escolha do método de crossover exerce maior influência sobre o tempo de execução do que o método de seleção, conforme também relatado por outros autores na literatura.
3. Aptidão (Fitness): A avaliação da aptidão foi validada com base em critérios objetivos como cor de fundo, temperatura do ambiente e características físicas. Logs detalhados foram gerados para comparar os resultados com os parâmetros estabelecidos.

4. Interação com o Ambiente: Foram realizados testes independentes para cada parâmetro ambiental, a fim de garantir que o sistema respondia adequadamente às mudanças no ambiente e gerava personagens alinhados às condições simuladas.

6.1 Comentários sobre a Implementação

Durante o desenvolvimento, foi identificado que o programa apresenta lentidão quando a população é muito grande, especialmente devido à renderização simultânea dos personagens. Além disso, ao executar mais de duas gerações simultâneas, o desempenho é ainda mais afetado. Para mitigar esses problemas, estão sendo estudados métodos de otimização no Unity, como técnicas de renderização e manipulação de objetos.

6.2 Análise de Resultados e Visualizações

Para analisar e comparar as gerações entre si e poder cruzar os resultados com os artigos estudados, foram geradas 9 populações simultâneas partindo de uma mesma população inicial, cada uma utilizando uma combinação distinta de método de seleção de pais (Roleta, Rank-Based ou Torneio) e método de crossover (Single Point, Double Point ou Scattered).

Inicialmente as populações teste possuíam 500 membros analisados ao longo de 500 gerações, porém foi notado que após a centésima geração, não haviam mudanças significativas, com a maioria das populações se mantendo igual ou estável, dessa forma foi estabelecido um limite de 100 gerações.

Com uma população de 500 membros ao longo de **100 gerações**, cada experimento gerou um total de **50.000 indivíduos** por combinação de métodos.

Cada log gerado continha, por personagem:

- Fitness individual da população;
- Velocidade do personagem;
- Quantidade de “fur” (relacionado ao calor corporal);
- Energia e visão;
- Altura das pernas e dos braços;
- Cor da pele, cabelo, olhos, camisa, mangas e boca (em hexadecimal);

- Temperatura ambiente e cor do solo (em hexadecimal);
- Taxas de mutação por atributo: física (fur, speed, energy, vision), visual (cores) e estrutural (altura dos membros);
- Um resumo textual dos componentes visuais (como olhos, boca, cabelo, torso, etc.) aplicados a cada personagem.

Esses dados foram exportados e analisados via Python na plataforma *Kaggle*.

6.2.1 Análises de Fitness

Para entender como cada método de seleção e crossover impacta o desempenho da população ao longo do tempo, foram analisados os valores de fitness médio em cada uma das 9 gerações. Como todas as populações começaram da mesma base, foi possível comparar diretamente a evolução de cada combinação de estratégias.

A fitness se trata da aptidão de um personagem no ambiente, ou seja, o quão bem adaptado e propenso a sobreviver e reproduzir esse personagem está. Nesse projeto, a fitness é formada por três fatores essenciais, sendo esses a camuflagem (semelhança da cor de pele do boneco com o fundo), a visão (personagens com pupilas maiores ou sem pupilas se adequam melhor em ambientes com pouca iluminação) e a quantidade de pelos(ambientes mais frios favorecem orelhas e rabos). Dessa forma a fitness máxima de um personagem é 3.

A análise foi repetida cinco vezes: quatro execuções utilizaram o mesmo fundo com coloração semelhante, enquanto uma quinta foi realizada com uma cor de fundo distinta, com o objetivo de investigar possíveis padrões relacionados ao ambiente. Em todas as execuções, foram mantidas constantes a taxa de aleatoriedade em 0,01 (1%), a iluminação em 0 e o calor ambiente também em 0, garantindo condições controladas para a comparação dos resultados.

Primeiramente foi analisada a fitness média das populações por execução, foram executadas 5 simulações contendo 9 populações cada. O objetivo foi verificar se os resultados obtidos seriam coerentes com o esperado e consistentes entre si. Os resultados dessas simulações se encontram nas figuras 11 à 15.

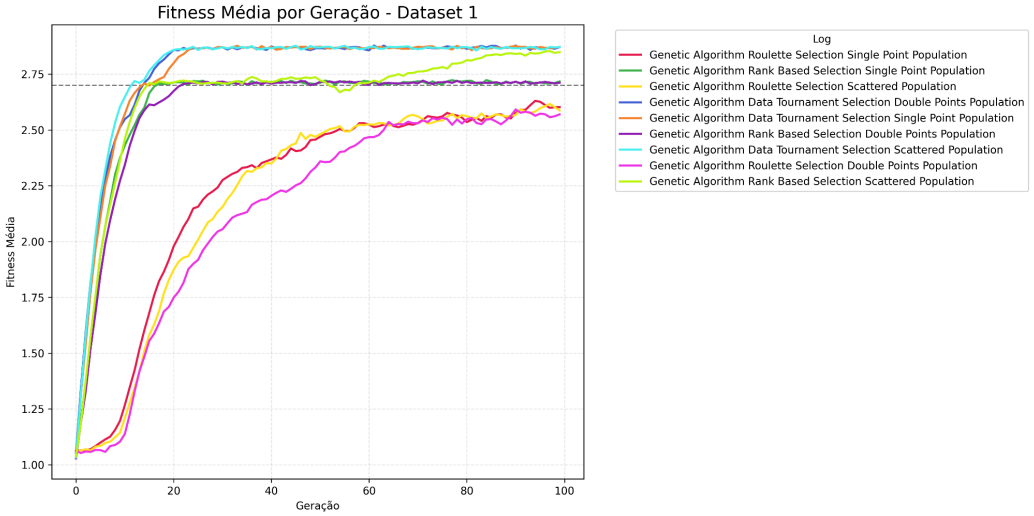

figura 11- Evolução da fitness média da simulação 1 nas populações utilizando os 3 métodos de seleção com os 3 métodos de crossover.

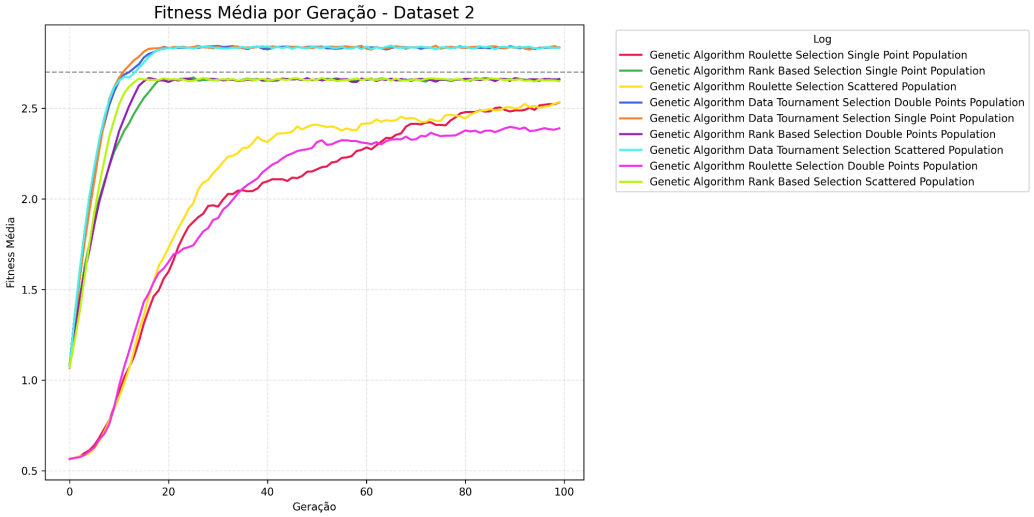

figura 12- Evolução da fitness média da simulação 2 nas populações utilizando os 3 métodos de seleção com os 3 métodos de crossover.

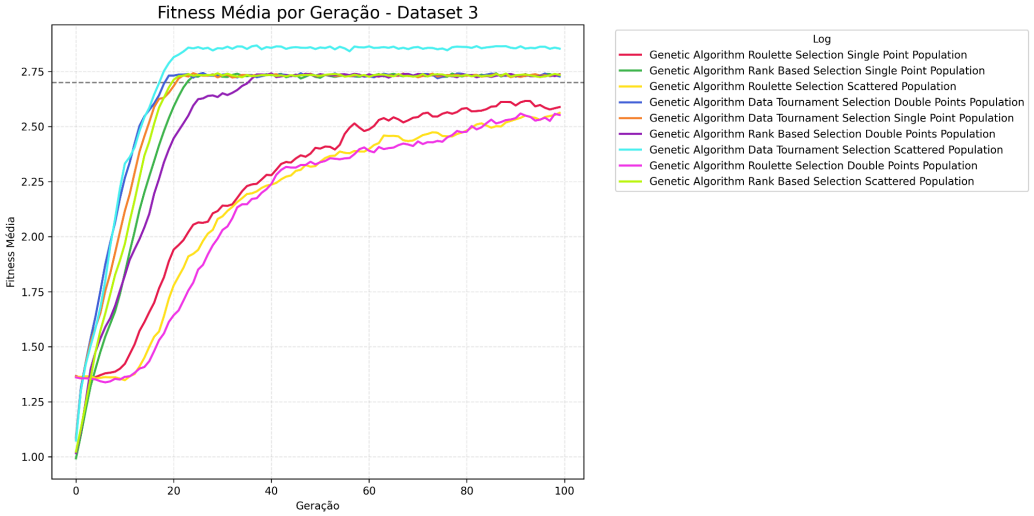

figura 13 - Evolução da fitness média da simulação 3 nas populações utilizando os 3 métodos de seleção com os 3 métodos de crossover.

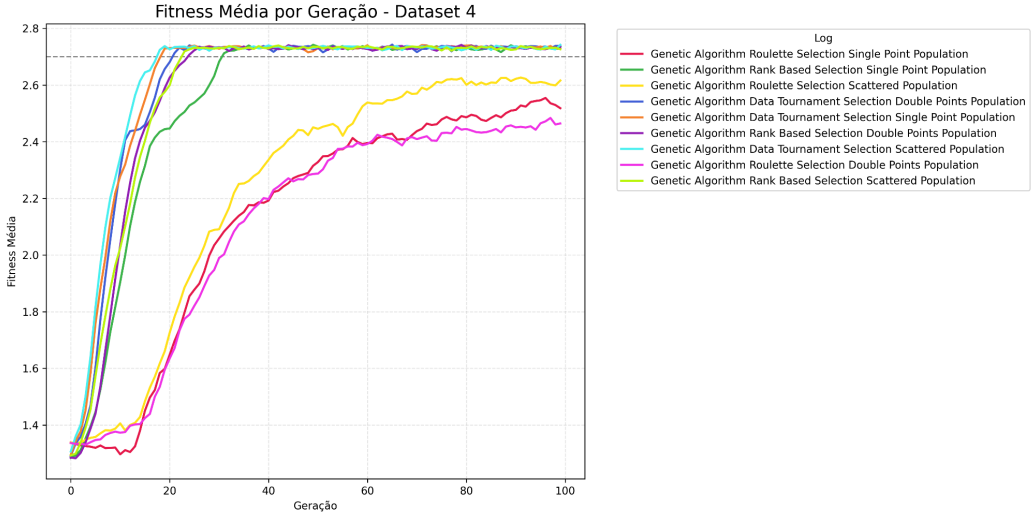

figura 14 - Evolução da fitness média da simulação 4 nas populações utilizando

os 3 métodos de seleção com os 3 métodos de crossover.

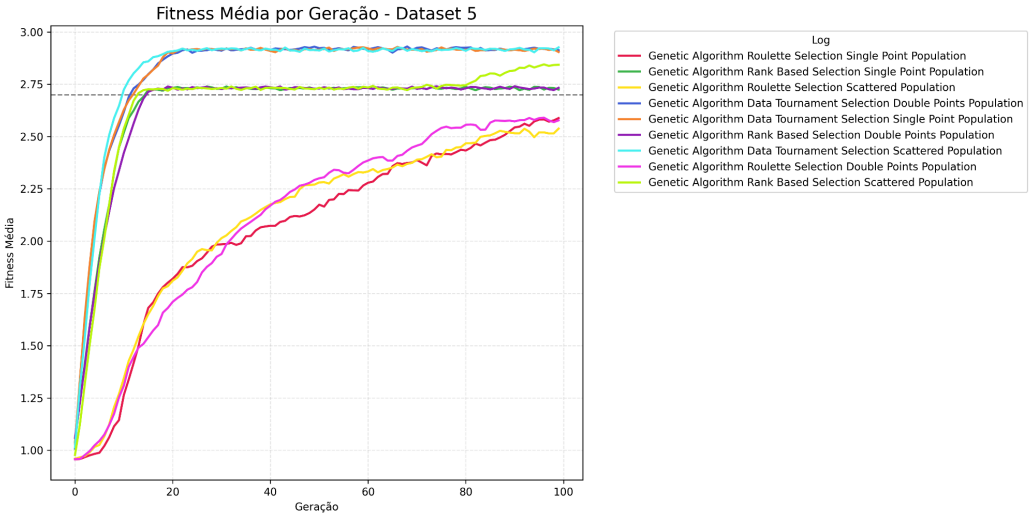

figura 15 - Evolução da fitness média da simulação 5 nas populações utilizando os 3 métodos de seleção com os 3 métodos de crossover.

Apesar das variações causadas pela natureza estocástica do algoritmo,como seleção e mutações aleatórias, é possível perceber que todas as curvas seguem um padrão bem definido. Em todos os gráficos, as populações começam com fitness médio mais baixo, apresentam uma subida rápida nas primeiras gerações,depois estabilizando em um valor próximo ao ótimo. Mesmo com os diferentes métodos de seleção e crossover, o comportamento geral se manteve consistente entre os cinco testes, mostrando que o algoritmo tende a evoluir de forma previsível dentro das mesmas condições.

As gerações cuja seleção parental se baseava em torneio apresentaram convergência mais rápida e atingiram fitness médios mais altos, enquanto as que usaram roleta tiveram desempenho inferior, independentemente do tipo de crossover utilizado. Esses resultados concordam com o estudado por Goldberg e Deb (1991), que mostraram que a roleta tem uma pressão seletiva instável e menos eficaz ao longo das gerações. Já a seleção por torneio mantém uma pressão seletiva consistente, favorecendo soluções de melhor desempenho com maior estabilidade.

Analisando mais detalhadamente os três métodos de melhor fitness (Torneio scattered, torneio single point e torneio double point) foi possível perceber que seguem o padrão de fitness similar. Isso possivelmente se dá pelo fato de que o cálculo de fitness do projeto não é complexo, se baseando na

soma de três fatores (temperatura, cor e visão); e quando se trata de um ambiente imutável, a taxa de convergência após alcançada não parece mudar drasticamente.

Em termos numéricos, podemos ver a fitness média de todas as gerações na figura 16. Os personagens cujo valor mais se aproxima de 3 (máxima estipulada) são os mais bem adaptados e propensos a sobreviver.

	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Média Final
Torneio + Scattered	2.872	2.834	2.854	2.742	2.926	2.846
Torneio + Double Point	2.871	2.836	2.727	2.738	2.913	2.817
Torneio + Single Point	2.871	2.834	2.738	2.729	2.904	2.815
Ranked + Scattered	2.849	2.65	2.734	2.726	2.843	2.76
Ranked + Double Point	2.711	2.662	2.736	2.734	2.733	2.715
Ranked + Single Point	2.717	2.656	2.736	2.738	2.725	2.714
Roleta + Scattered	2.587	2.534	2.562	2.616	2.538	2.567
Roleta + Single Point	2.602	2.532	2.588	2.518	2.588	2.566
Roleta + Double Point	2.57	2.39	2.553	2.464	2.577	2.511

figura 16 - Tabela contendo a fitness média de todas as populações nas 5 simulações executadas.

Dessa forma podemos analisar que a maior média final foi obtida com a escolha Torneio +Scattered.

Os métodos de maior fitness se mantiveram constantes com a escolha do torneio, dada a sua natureza competitiva que prioriza melhores resultados.

Observando a variedade genética, as populações que usaram seleção por roleta apresentaram os maiores desvios padrão de fitness, o que indica uma diversidade maior entre os indivíduos até o final da simulação. Já as populações com seleção por torneio foram as que mais rapidamente se tornaram homogêneas, com indivíduos muito parecidos entre si, o que também está de acordo com o esperado. Isso reforça o que foi descrito por Goldberg e Deb (1991), que apontam que a roleta pode gerar oscilações e favorecer indivíduos aleatórios. A figura 17 exemplifica em forma de gráfico a média de desvio padrão correspondente a cada método de geração.

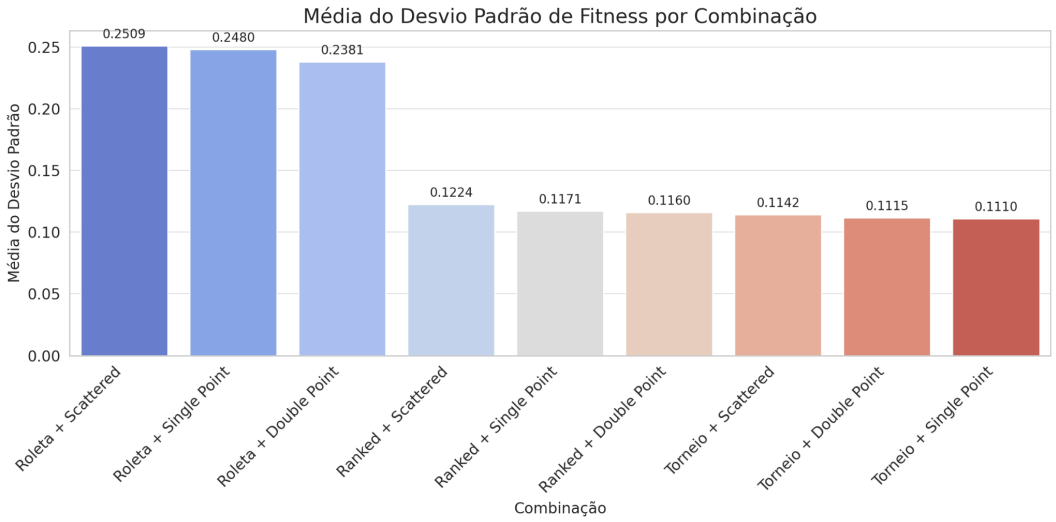

figura 17 - Tabela contendo a média do desvio padrão de todas as populações nas 5 simulações executadas.

6.2.2 Análise dos personagens de maior fitness

Ao analisar as populações mostradas na sessão anterior, que originaram os indivíduos com maior valor fitness individualmente , observamos que a maioria deles pertencem à combinação de Rank + Single Point e Rank + Scattered. Esse resultado evidencia que, embora a média de fitness considere a soma total dividida pelo número de indivíduos, ela não necessariamente reflete a presença de indivíduos altamente otimizados.Além disso, foi possível perceber que em quase todos os métodos foi possível alcançar um indivíduo ótimo em algum momento.

Para cada simulação, foram gerados gráficos mostrando o indivíduo de maior fitness por população, ou seja, para cada uma das 5 execuções, foi selecionado o melhor indivíduo por combinação de métodos de crossover e -seleção parental.

Os gráficos das simulações se encontram nas imagens 18 à 22.

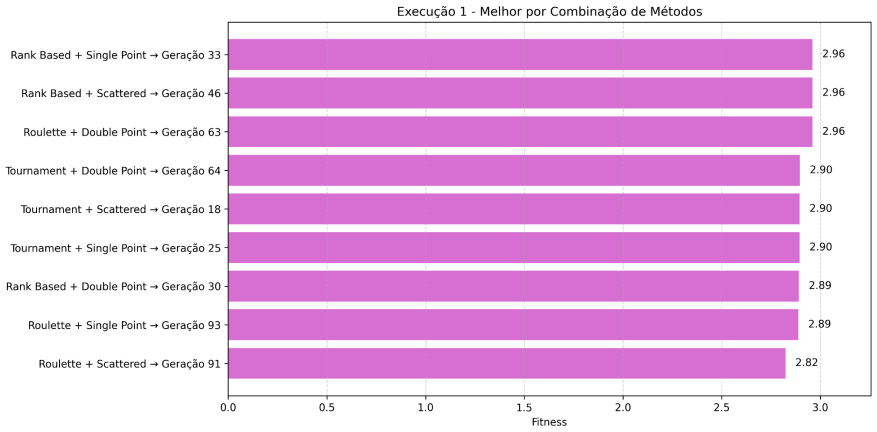

figura 18 - Gráfico contendo a fitness e geração correspondente ao melhor indivíduo de cada população na simulação 1

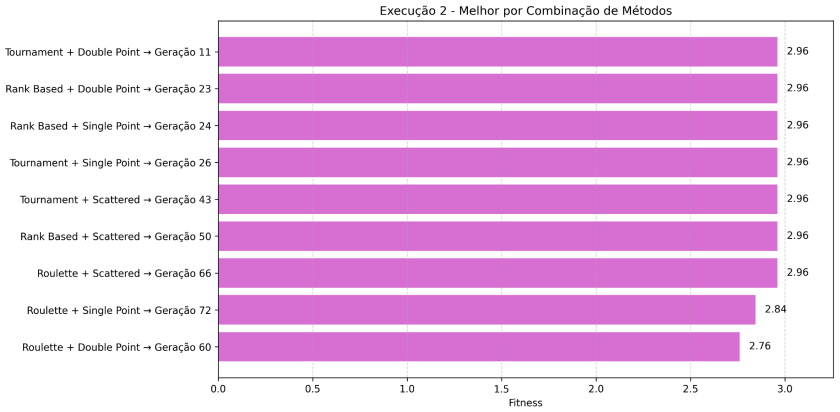

figura 19 - Gráfico contendo a fitness e geração correspondente ao melhor indivíduo de cada população na simulação 2

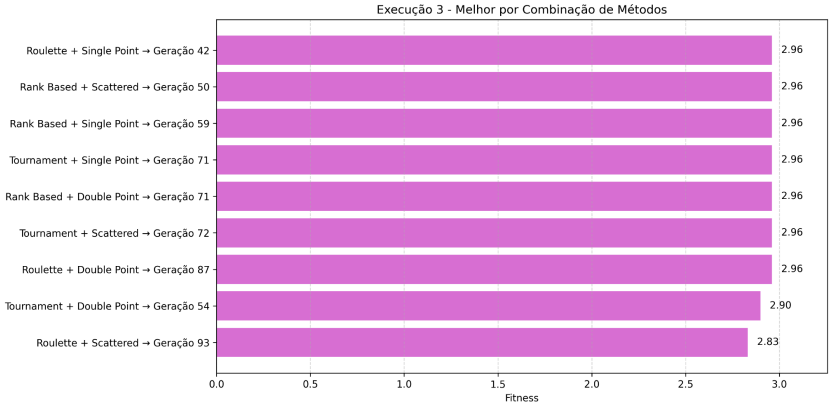

figura 20 - Gráfico contendo a fitness e geração correspondente ao melhor indivíduo de cada população na simulação 3

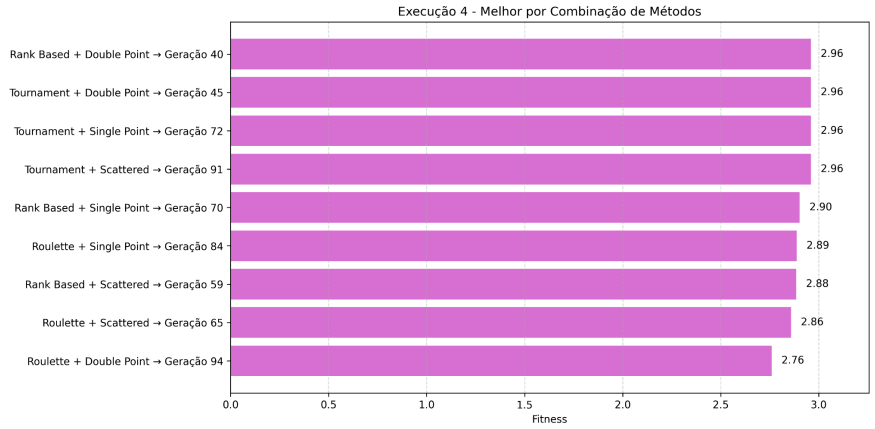

figura 21 - Gráfico contendo a fitness e geração correspondente ao melhor indivíduo de cada população na simulação 4

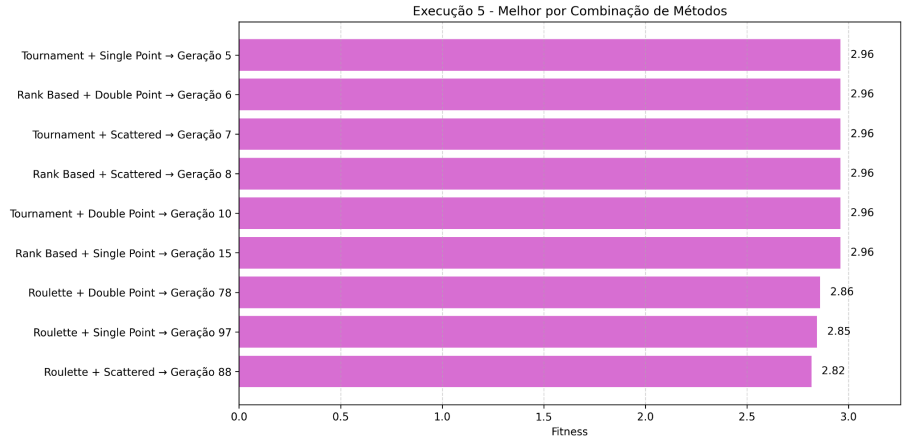

figura 22 - Gráfico contendo a fitness e geração correspondente ao melhor indivíduo de cada população na simulação 5

A partir da análise de seus dados genéticos e da sua reconstrução no ambiente do Unity, é possível verificar que seus atributos estão fortemente alinhados com as condições ambientais simuladas.Essas reconstruções podem ser observadas na imagem 23.

figura 23 - Representação visual dos personagem que obtiveram melhor fitness em cada execução

Além disso é possível observar que todos possuem cores e atributos adequados ao ambiente, como pode ser verificado na tabela de atributos apresentada no apêndice A e na imagem 24.

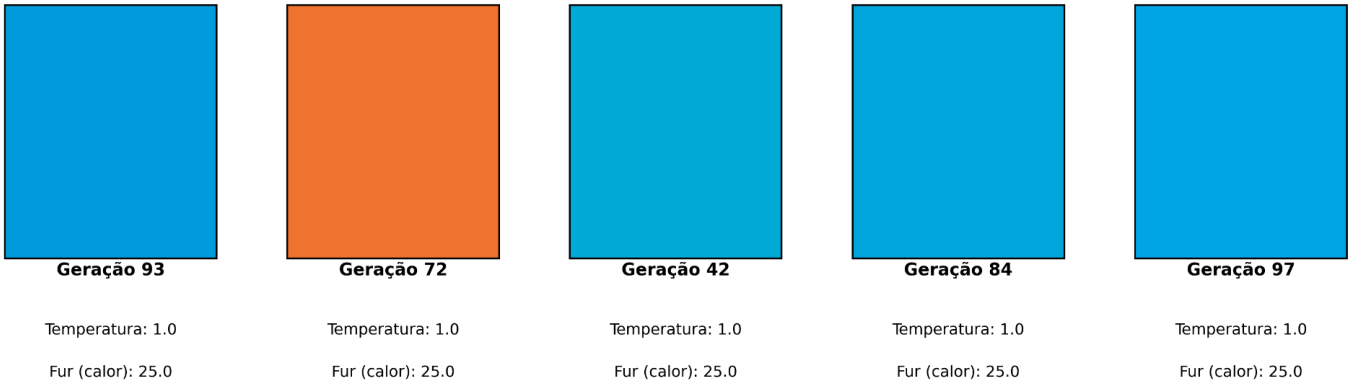

figura 24- Imagens do ambiente da geração com informações sobre cor, temperatura e calor.

6.2.3 Análises de cor

Para avaliar a variação de cor durante as gerações, foram criados gráficos que mostrassem a média de cara geração juntamente da cor do ambiente em dado momento.Cada gráfico apresenta a variação de cor da população durante a geração, a cor do ambiente e uma linha representando a convergência cromática de ambos os dados. Quanto mais abaixo a linha, mais similares são as cores.

Os resultados, que podem ser visualizados na imagem 24, foram condizentes com os dados obtidos na análise de fitness, uma vez que a variação de cor está diretamente relacionada com a adaptação dos personagens ao ambiente.

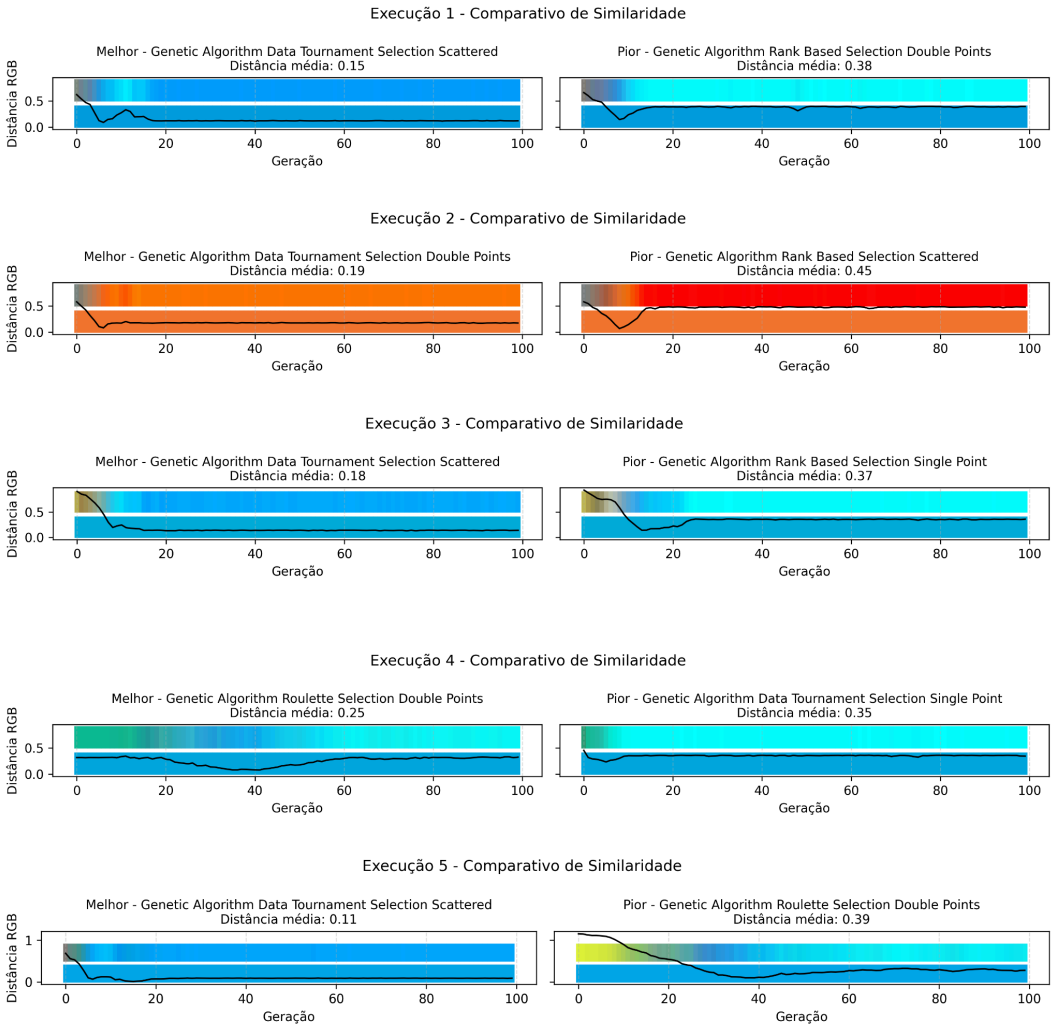

figura 25 - Gráficos comparando a cor do ambiente com a cor média da população ao longo das gerações na simulação para cada uma das 5 simulações.

7. Considerações Finais

Esse projeto foi uma oportunidade de unir teoria e prática de forma concreta. Trabalhar com algoritmos genéticos me permitiu não só entender melhor os fundamentos por trás de seleção, crossover e mutação, como também observar, de forma visual e interativa, os impactos reais de cada estratégia ao longo das gerações. A ferramenta criada acabou servindo tanto como protótipo funcional quanto como uma plataforma de experimentação, permitindo testar hipóteses diretamente no ambiente da Unity, com o suporte de análise estatística

via scripts em Python.

Com base nos testes realizados e nos artigos estudados, foi possível perceber padrões interessantes. Em relação à variação genética, a seleção por **roleta** se mostrou determinante para uma população mais diversa, ao passo que a seleção por **torneio**, apresentou melhor desempenho médio e maior estabilidade entre execuções — algo já apontado por Gabriel (2015) e reforçado por Rawat et al. (2022).

Já os métodos de crossover se mostraram ainda mais determinantes na taxa de convergência: o **double point**, como sugerido por Park e Park (2021), teve uma performance bastante eficiente, enquanto o **scattered** se destacou nas primeiras gerações por manter mais diversidade, como defendido por Bye et al. (2020). Isso se refletiu não só nas curvas de fitness, mas também na variação de cor e nos gráficos de convergência cromática ao longo do tempo.

Um ponto que vale destacar é que, embora a média de fitness tenha sido maior em populações com **Torneio + Scattered**, os indivíduos com melhor fitness isolado geralmente vieram da combinação **Rank + Single Point** e **Rank + Scattered**. Isso reforça que uma média alta não garante um indivíduo ótimo — algo importante dependendo do objetivo da aplicação.

Em termos de variedade genética, a combinação que gerou maior desvio padrão foi **Roleta + Scattered.**

Na prática, o projeto também trouxe aprendizados importantes sobre escopo e desempenho. A ideia de permitir múltiplos métodos de seleção e crossover, somada à possibilidade de rodar várias populações ao mesmo tempo, trouxe um nível de complexidade que acabou exigindo muito tempo em polimento e depuração. Se eu fosse começar de novo, com certeza definiria um escopo mais contido no início — talvez limitando a geração a uma população única e introduzindo variações aos poucos. Além disso, teria investido mais cedo em técnicas de otimização no Unity, como *object pooling* e controle de renderização, pois ficou claro que o desempenho cai muito com populações grandes.

Para versões futuras, seria interessante explorar:

- Novos métodos de mutação e estratégias como elitismo ou reprodução

com penalidade por similaridade (como sugerido por Goldberg e Deb, 1991);

- Adição de comportamentos mais complexos, como “sobrevivência do mais forte”, busca por recursos ou movimentação adaptativa;

- Um sistema de ambiente mais dinâmico, com ciclos de calor e luz, que possam exigir respostas evolutivas mais sofisticadas — algo que se aproxima dos desafios descritos por Shan (2021) e Kushner (2008) em projetos como Spore.

Mesmo com suas limitações, acredito que o sistema desenvolvido cumpre bem seu papel como uma ferramenta de visualização e experimentação com algoritmos genéticos aplicados à criação de personagens. Ele oferece um ponto de partida para simulações mais complexas e também serve como suporte para ensino ou testes comparativos. Mais do que só gerar personagens diferentes, ele permite refletir sobre como a evolução pode ser controlada, observada e ajustada — o que, no fim, era justamente a ideia central do trabalho.

8. Referências bibliográficas

ABULAIL, Rawan Nassri. Premature Avoidance in Genetic Algorithm using Dynamic Mutation Probability. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, v. 16, n. 1, p. 31–47, 2025. Disponível em: https://jowua.com/wp-content/uploads/2025/04/2025.I1.031.pdf. Acesso em: 20 abr. 2025.

AMBUEHL, Nathan. Investigating Genetic Algorithm Optimization Techniques in Video Games. 2017. Trabalho de conclusão de curso (Graduação em Ciência da Computação) – East Tennessee State University. Disponível em: https://dc.etsu.edu/honors/748. Acesso em: 14 jul. 2025.

BACH, Erik; MADSEN, Anders. *Procedural Character Generation: Implementing Reference Fitting and Principal Components Analysis*. MSc Thesis. Aalborg Universitet, 2007.

BYE, Robin; GRIBBESTAD, Magnus; CHANDRA, Ramesh; OSEN, Ottar. A Comparison of GA Crossover and Mutation Methods for the Traveling Salesman Problem. In: *Proceedings of the International Conference on Computational Science*, 2020. p. 529–542. DOI: https://doi.org/10.1007/978-981-15-6067-5_60.

COSTA JUNIOR, A. R. *Ferramenta para Geração Procedimental de Níveis em Jogos Infinitos*. Universidade Federal de Pernambuco, Centro de Informática, 2023. Disponível em: https://www.cin.ufpe.br/~tg/2013-2/arcj.pdf. Acesso em: 27 dez. 2024.

COUTINHO, Flávio; CHAIMOWICZ, Luiz. On the Challenges of Generating Pixel Art Character Sprites Using GANs. In: *Proceedings of the 18th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2022. Disponível em: https://ojs.aaai.org/index.php/AIIDE/article/view/21951. Acesso em: 14 jul. 2025.

ECCLES, Vanya. *Procedurally Generated Background Characters*. 2017. 94 f. Dissertação (M.Sc. em Ciência da Computação – Interactive Entertainment Technology) – University of Dublin, Trinity College, Dublin, 2017. Disponível em: https://publications.scss.tcd.ie/theses/diss/2017/TCD-SCSS-DISSERTATION-2017-036.pdf. Acesso em: 15 jul. 2025.

FERREIRA, Felipe Vieira. *Aplicação do algoritmo neuroevolutivo NEAT para direção autônoma*. 2025. 62 f. Projeto Final de Graduação (Graduação em Ciência da Computação) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2025. Disponível em: https://projgrad.inf.puc-rio.br/uploads/20251_1711087/PF2_relatorio_1711087.pdf . Acesso em: 15 jul. 2025.

GABRIEL, J. S. Uma abordagem comparativa entre técnicas de seleção em algoritmos genéticos para otimização de soluções. In: *SBGAMES*, 2015. Disponível em: https://edirlei.com/papers/Gabriel_SBGames_2015.pdf. Acesso em: 2 jan. 2025.

GAISBAUER, Wolfgang; HLAVACS, Helmut. Procedural Attack! Procedural Generation for Populated Virtual Cities: A Survey. *International Journal of Serious Games*, v. 4, n. 2, p. 19–29, 2017. DOI: https://doi.org/10.17083/ijsg.v4i2.161.

GOLDBERG, David E.; DEB, Kalyanmoy. A comparative analysis of selection schemes used in genetic algorithms. 1991. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/B9780080506845500082. Acesso em: 19 abr. 2025.

KHALIFA, Ahmed; BONTRAGER, Philip; EARLE, Sam; TOGELIUS, Julian. PCGRL: Procedural Content Generation via Reinforcement Learning. In: *Proceedings of the 16th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2020. Disponível em: https://arxiv.org/abs/2001.09212. Acesso em: 14 jul. 2025.

KUSHNER, David. Engineering Spore. *IEEE Spectrum*, v. 45, n. 9, p. 36–40, set. 2008. DOI: https://doi.org/10.1109/MSPEC.2008.4607929.

MALEKI, Mahdi Farrokhi; ZHAO, Richard. Procedural Content Generation in Games: A Survey with Insights on Emerging LLM Integration. *arXiv preprint* arXiv:2410.15644, 2024. Disponível em: https://arxiv.org/abs/2410.15644. Acesso em: 14 jul. 2025.

PARK, Sung-Bae; PARK, Jung-Ho. A Comparative Study of Selection and Crossover Methods in Genetic Algorithms. *Computer Science & Information Technology (CS & IT)*, v. 10, p. 1–11, 2021. Disponível em: https://aircconline.com/csit/papers/vol10/csit101101.pdf. Acesso em: 14 jul. 2025.

RAWAT, Bibek; DUWAL, Dipesh; PHUYAL, Sagar; PANT, Aparana. A Comparative Review Between Various Selection Techniques in Genetic Algorithm for Finding Optimal Solutions. *International Journal of Computer Sciences and Engineering*, v. 10, p. 15–22, 2022. DOI: https://doi.org/10.26438/ijcse/v10i10.1522.

SHAN, Yuanqi. *A Procedural Character Generation System*. Aaltodoc, 2021. Disponível em: https://aaltodoc.aalto.fi/handle/123456789/108347. Acesso em: 27 dez. 2024.

SOON, G. K.; GUAN, T. T.; ON, C. K.; ALFRED, R.; ANTHONY, P. A comparison on the performance of crossover techniques in video game. In: *IEEE International*

Conference on Control System, Computing and Engineering, Penang, Malaysia, 2013. p. 493–498. DOI: https://doi.org/10.1109/ICCSCE.2013.6720015.

SUMMERVILLE, Adam et al. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games*, v. 10, n. 3, p. 257–270, 2018. DOI: https://doi.org/10.1109/TG.2018.2846639.

TAIT, E. R.; NELSON, I. L. Nonscalability and generating digital outer space natures in No Man's Sky. *Environment and Planning E: Nature and Space*, v. 5, n. 2, p. 694–718, 2022. DOI: https://doi.org/10.1177/25148486211000746.

TOGELIUS, Julian; YANNAKAKIS, Georgios; STANLEY, Kenneth. Search-based procedural content generation. In: *Proceedings of EvoApplications*. Springer LNCS, 2010.

UMBARAKAR, A. J.; SHETH, P. D. Crossover operators in genetic algorithms: a review. *ICTACT Journal on Soft Computing*, v. 6, n. 1, p. 1083–1092, 2015. Disponível em: https://ictactjournals.in/paper/IJSC_V6_I1_paper_4_pp_1083_1092.pdf. Acesso em: 18 abr. 2025.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. *Geração procedural de personagens para jogos: Uma abordagem prática utilizando algoritmos genéticos*. 2020. Disponível em: https://lume.ufrgs.br/handle/10183/224871. Acesso em: 2 jan. 2025.

VIANA, Breno M. F. et al. Feasible–Infeasible Two-Population Genetic Algorithm to Evolve Dungeon Levels with Dependencies in Barrier Mechanics. *Applied Soft Computing*, v. 119, p. 108586, 2022. DOI: https://doi.org/10.1016/j.asoc.2022.108586.

UNITY TECHNOLOGIES. *ScriptableObject*. In: Unity User Manual (version 6.1). Disponível em: https://docs.unity3d.com/6000.1/Documentation/Manual/class-ScriptableObject.html. Acesso em: 15 jul. 2025.

UNITY TECHNOLOGIES. Transform (Unity Scripting API). Disponível em: https://docs.unity3d.com/ScriptReference/Transform.html. Acesso em: 15 jul. 2025.

9. Apêndice A

tabela contendo todos os atributos utilizados na base de dados:

Sprite	Parte do Corpo	Fur	Vision
	Arm	0	0
	Eye	0	4
	Eye	0	0
	Eye	0	1
	Eye	0	2
	Eye	0	9
	Eye	0	8
	Eye	0	7
	Eye	0	6

Sprite	Parte do Corpo	Fur	Vision
	Eye	0	5
	Eye	0	4
	Eye	0	10
	Eye	0	3
	Foot	0	0
	Hair	7	0
	Hair	4	0
	Hair	10	0
	Hair	2	0

Sprite	Parte do Corpo	Fur	Vision
	Hair	3	0
	Hair	0	0
	Hair	5	0
	Hand	0	0
	Head	0	0
	Head	0	0
	Head	0	0
	Head	0	0
	Head	0	0

Sprite	Parte do Corpo	Fur	Vision
	Head	0	0
	Head	0	0
	Head	0	0
	Head	0	0
	Head	5	0
	Head	0	0
	Leg	0	0
	Mouth	0	0
	Mouth	0	0

Sprite	Parte do Corpo	Fur	Vision
	Mouth	0	0
	Mouth	0	0
	Mouth	0	0
	Mouth	0	0
	Mouth	0	0
	Mouth	0	0
	Mouth	0	0
	Mouth	0	0
	Sleeve	0	0

Sprite	Parte do Corpo	Fur	Vision
	Sleeve	0	0
	Tail	0	0
	Tail	10	0
	Torso	0	0
	Torso	0	0
	Torso	0	0