UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Lucas Schürer

GERAÇÃO PROCEDURAL DE INIMIGOS ADAPTATIVOS EM JOGOS UTILIZANDO ALGORITMOS GENÉTICOS

Santa Maria, RS
2023

Lucas Schürer

GERAÇÃO PROCEDURAL DE INIMIGOS ADAPTATIVOS EM JOGOS UTILIZANDO ALGORITMOS GENÉTICOS

Monografia apresentada ao Curso de Graduação em Sistemas de Informação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Sistemas de Informação**.

Orientador: Prof. Joaquim Vinicius de Carvalho de Assunção

Santa Maria, RS
2023

RESUMO

GERAÇÃO PROCEDURAL DE INIMIGOS ADAPTATIVOS EM JOGOS UTILIZANDO ALGORITMOS GENÉTICOS

AUTOR: Lucas Schürer
Orientador: Joaquim Vinicius de Carvalho de Assunção

Algoritmos para geração procedural de conteúdo são utilizados extensivamente pela indústria de jogos, servindo para aumentar a vida útil do produto e proporcionar uma melhor experiência para o usuário. No entanto, a geração procedural de inimigos ainda é uma prática pouco comum. Esta monografia, através da implementação de um jogo baseado em *waves*, busca avaliar o comportamento de inimigos adaptativos gerados proceduralmente.

Palavras-chave: Geração procedural. Jogos. Algoritmos genéticos. Geração procedural de conteúdo.

ABSTRACT

PROCEDURAL GENERATION OF ADAPTIVE ENEMIES IN GAMES USING GENETIC ALGORITHMS

AUTHOR: Lucas Schürer
ADVISOR: Joaquim Vinicius de Carvalho de Assunção

Algorithms for procedural content generation are widely used in the gaming industry, aiming to increase the product's lifespan and provide a better user experience. However, procedurally generating enemies is not as common. This work, through the implementation of a wave-based game, seeks to evaluate the behaviour of adaptively generated enemies.

Keywords: Procedural generation. Games. Genetic algorithms. Procedural content generation

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

IA Inteligência Artificial

GA *Genetic Algorithm*

PCG *Procedural Content Generation*

NPC *Non-playable character*

$ARPG$ *Action role-playing game*

FSM *Finite state machine*

SUMÁRIO

1 INTRODUÇÃO

A capacidade de um jogo poder ser experienciado de múltiplas maneiras diferentes é explorada de maneira extensiva atualmente. Jogos como *Minecraft*, *Divinity: Original Sin*, *League of Legends* e *FINAL FANTASY XIV Online* podem fornecer várias horas de conteúdo, seja por sua capacidade de criar mundos diferentes, as diversas maneiras de se finalizar o jogo ou seu aspecto competitivo e social.

Através da geração procedural de conteúdo (PCG), jogos podem personalizar a experiência para cada jogador e aumentar o fator de rejogabilidade - tempo em que uma pessoa pode se divertir com um jogo até que se torne entediante (KRALL, 2012).

Assim, PCG é usada de maneira ampla na indústria de jogos, possuindo por volta de 2000 jogos marcados com a *tag* geração procedural na *Steam* na data desta monografia. No entanto, poucos jogos realizam a implementação de inimigos gerados proceduralmente, com essas técnicas sendo aplicadas na geração de mapas e itens.

Inimigos são um dos elementos presentes em quase todos os jogos, incluindo uma ameaça e servindo como forma de dificultar o jogador de alcançar seu objetivo. Mesmo assim, como jogador, estamos limitados a experienciar os mesmos inimigos em ambientes diferentes.

Esta monografia avalia a capacidade de adaptação de inimigos gerados proceduralmente através de algoritmos genéticos, buscando entender o motivo pelo qual não é usado amplamente na indústria. Para isso, um jogo foi desenvolvido, com agentes sendo colocados em cenário de teste com outros agentes, com seu comportamento sendo observado diante das diferentes situações.

No capítulo 2 passaremos por conceitos importantes a respeito do que é um jogo, para então entendermos rejogabilidade. Através dessa discussão, podemos entender a importância de PCG e sua aplicação na indústria e em pesquisas acadêmicas realizadas nesse campo. Também falaremos sobre outras pesquisas com um foco na geração procedural de inimigos, onde discutiremos o avanço obtido nessa área.

Depois, no capítulo 4, entenderemos o que são algoritmos genéticos, explicando seus conceitos, etapas e utilizações. Exemplos serão dados para melhor visualização e que servirão como uma base para entendermos a aplicação de GA na geração procedural dos inimigos no jogo desenvolvido para esta monografia.

Também veremos a implementação utilizada para o comportamento dos agentes, a qual foi criada através de uma máquina de estados, no capítulo 5.

No capítulo 6 entramos em detalhes a respeito do desenvolvimento do jogo utilizado para a realização de testes, mostrando seus sistemas e peculiaridades geradas para a utilização de algoritmos genéticos, bem como as dificuldades para implementação - fator que pode influenciar na decisão dos desenvolvedores de não utilizar essa técnica.

Por fim, no capítulo 7, discutimos os dados coletados nos cenários de testes conduzidos com os inimigos, buscando analisar seus comportamentos e suas reações em diferentes condições. Dessa forma, poderemos compreender melhor se a não utilização desse método é devido a sua baixa eficácia, onde inimigos não apresentam uma mudança significativa no comportamento que compense o esforço para a implementação da técnica.

2 TRABALHOS RELACIONADOS

Em *Homo Ludens* (HUIZINGA, 1949), Huizinga define o ato de jogar como uma necessidade, realizada por humanos e animais e precedendo a sociedade. Em sua obra, introduziu o conceito do círculo mágico, local onde o ato de jogar é realizado. No círculo mágico, as regras do mundo normal são suspensas, substituídas pelas regras do jogo. Para ele, uma das características inerentes de jogar é a repetição, inclusa em quase todas as formas de jogar.

Salen e Zimmerman em *Rules of Play* (SALEN, 2004) fornecem um contraponto a abordagem de Huizinga, descrevendo-a como muito generalista, não realizando distinção entre o ato de jogar e jogos. Mesmo assim, reconhecem que *Homo Ludens* identificou conceitos abstratos importantes para a definição de jogos. Expandiram o conceito inicialmente proposto por Huizinga, a respeito do círculo mágico, trazendo-o para um contexto de jogos e não unicamente sobre o ato de jogar.

Em sua versão do círculo mágico, Salen e Zimmerman, o círculo mágico define um espaço capaz de definir regras que podem ser vistas como formas ineficientes de atingir o objetivo do jogo, mas necessárias para o seu funcionamento. Mesmo com esse poder autoritário, o circulo mágico é frágil, necessitando de constante manutenção para se manter, atingida através de boas decisões de *design*.

A importância da rejogabilidade, ou *replayability*, é descrita por Salen e Zimmerman, com um exemplo de jogos de tabuleiro, onde a possibilidade de ser jogado múltiplas vezes é crucial. Krall e Menzies (KRALL, 2012) também fazem menção ao círculo mágico, aprofundando nos termos *playability* - a capacidade de uma atividade ser divertida ou não - e *replayability* - o quanto uma pessoa pode desfrutar de uma atividade até ela deixar de ser divertida. Para Krall e Menzies, jogos precisam possuir essas duas características.

Para aumentar o *replay value*, ou rejogabilidade, diversos jogos utilizam de estratégias para aplicar PCG. Este tipo de geração procedural engloba todos os aspectos de um jogo que afetam o *gameplay*, mas que não se enquadram no comportamento de NPCs ou alteram o funcionamento da *engine* utilizada (YANNAKAKIS; TOGELIUS, 2011). Yannakakis e Togelius levantam o questionamento do motivo da geração de conteúdo não ser utilizada de maneira mais ampla, mas, desde 2011, a indústria dos jogos presenciou um grande avanço neste tópico.

Jogos como *Minecraft* e *Deep Rock Galactic* utilizam da geração procedural de mapas para garantir uma melhor rejogabilidade, proporcionando um sentimento de descoberta mesmo depois de múltiplas jogatinas.

O subgênero de *Loot-Based ARPGs* aposta na repetição como parte fundamental de seu ciclo de jogo. Nesse subgênero, o uso da geração procedural de itens é extremamente importante. Em *Path of Exile*, é esperado que o mesmo conteúdo seja experien-

ciado diversas vezes, com a possibilidade de itens melhores, esses gerados de maneira procedural, serem adquiridos.

Mesmo com PCG sendo popular e amplamente usada, a geração procedural de inimigos é um assunto que foi pouco pesquisado na data de publicação desta monografia e pouco utilizada por jogos.

No jogo *Left 4 Dead*, um dos elementos do jogo é a presença da *AI Director*, responsável por controlar a dificuldade e experiência do jogador, aumentando ou diminuindo o número, frequência ou local de surgimento dos inimigos. Essa abordagem, mesmo que garanta cenários que estão reagindo as ações do jogador, não garante novas características aos inimigos.

Em *Middle-Earth: Shadow of Mordor*, inimigos reagem as ações do jogador, ganhando características provenientes dessas reações com o intuito de aumentar a imersão.

No artigo *Procedural Enemy Generation through Parallel Evolutionary Algorithm* (PEREIRA; VIANA; TOLEDO, 2021), a geração procedural de inimigos com o propósito de alcançar um determinado nível de dificuldade é proposta, utilizando algoritmos evolutivos.

Buscando atualizar o comportamento de inimigos em tempo real durante uma partida, Font (FONT, 2012) alterou o comportamento dos agentes para deixar o jogo em um nível de dificuldade satisfatório. No entanto, Font não implementa nenhuma mudança em atributos dos inimigos, como vida, velocidade de movimento, ataque, etc...

Olsson (OLSSON, 2019) também busca atualizar o comportamento dos inimigos em tempo real, mas com um foco na adaptabilidade dos atributos. Em sua pesquisa, os inimigos gerados se comportaram mais aleatoriamente e com menos adaptabilidade quando os jogadores mudavam muito de estilo de jogo, fazendo com que os inimigos não se adaptassem.

As pesquisas de Olsson e Font contribuem para o estudo da geração procedural de inimigos em tempo real, implementando diferentes atributos e comportamentos. Esta monografia continua e expande o estudo na geração procedural de inimigos, implementando mudanças em atributos, comportamento e habilidades especiais de inimigos.

3 OBJETIVOS

Avaliar o impacto no comportamento proveniente da adaptação em diversos cenários, utilizando algoritmos genéticos para a geração procedural dos agentes.

Para isso, um jogo baseado em *waves* será desenvolvido, utilizando do ciclo de jogo característico do gênero para auxiliar na aplicação de algoritmos genéticos, com a relação de *waves* com gerações.

Ao fim desta monografia, os resultados obtidos serão detalhados, buscando entender a eficácia do método e suas dificuldades, fator que pode influenciar a baixa utilização em jogos.

4 ALGORITMOS GENÉTICOS

Descritos como uma solução para problemas de otimização em *Adaptation in Natural and Artificial Systems* (HOLLAND, 1992), algoritmos genéticos buscam replicar aspectos observados na natureza. Através da seleção de criaturas aptas e a aplicação de operadores genéticos como *crossover* e mutação, problemas com uma complexidade alta e que não demandam uma única solução correta são passíveis de serem resolvidos por algoritmos genéticos.

Algoritmos genéticos, mesmo que usados primariamente para problemas de otimização, também podem ser aplicados em outras áreas, como *machine learning*, processamento de imagens, redes neurais, economia e outros (MIRJALILI, 2019).

GAs, por conta de sua grande versatilidade, são extremamente úteis para a geração procedural de inimigos que precisam se adaptar ao estilo de jogo do jogador. Para produzir um comportamento mais natural na adaptação, uma certa variedade é desejada, característica que pode ser facilmente obtida ao utilizar de algoritmos genéticos. Além disso, o objetivo não é criar inimigos impossíveis de serem combatidos, portanto, não é necessária a preocupação em chegar a uma única solução ideal.

4.1 CONCEITOS

Algoritmos genéticos podem ser representados através de gerações, constituídas por uma população. A cada geração, o objetivo é alcançar uma maior *fitness*, valor que representa o quão bem as criaturas performaram no seu ambiente.

Entraremos posteriormente em mais detalhes a respeito de criaturas e *fitness*, mas, para uma melhor visualização dos conceitos apresentados anteriormente, podemos utilizar o seguinte exemplo: Em um jogo, a cada *wave* (geração), diversos inimigos (população) surgem com o propósito de impedir o avanço do jogador, destruindo-o (objetivo, que será usada para determinar a *fitness* de uma criatura). Dessa forma, a cada nova *wave*, inimigos que chegaram mais perto do objetivo serão selecionados, com suas características sendo passadas para inimigos de uma próxima *wave*.

O quadro 1 traça uma relação entre os conceitos de algoritmos genéticos abordados neste capítulo e sua possível representação em jogos.

Quadro 1 – Relação GAs com jogos.

GA	Jogo
População e criatura	Inimigos em um jogo
Geração	Rodada em uma partida
Cromossomos e genes	Inimigo possui características como forma de locomoção e ataque
Fitness	O quão perto o inimigo chegou de derrotar o jogador?
Seleção	Escolha dos inimigos que mais se aproximaram do objetivo
Crossover	Mescla das características dos inimigos escolhidos
Mutação	Randomizar características de alguns inimigos

4.1.1 Criatura, cromossomos e genes

A criatura é o nosso agente, responsável por interagir com o ambiente e possuindo características próprias, chamadas de cromossomos. Cromossomos, por sua vez, são compostos de genes.

Os genes de uma criatura formam um conjunto de características que determinam o seu comportamento em um ambiente específico. É importante destacar que um gene pode ser representado por qualquer tipo de dado, desde tipos mais primitivos, como binário ou *strings* até objetos. A presença de um gene específico pode influenciar em todos os outros genes do cromossomo, criando reações que podem ou não alterar o comportamento da criatura.

Para melhor visualizar criaturas, cromossomos e genes, iremos expandir o exemplo citado anteriormente, em 4.1. Os inimigos, com o objetivo de eliminar o jogador, possuem características distintas. Por exemplo, um inimigo pode voar e atacar o jogador de uma curta distância, enquanto um outro inimigo apenas anda, mas pode utilizar de ataques a distância.

Observando as características dos inimigos, podemos definir dois tipos de genes: método de locomoção e tipo de ataque. Com essa informação, também podemos construir o cromossomo das duas criaturas: [*voador, curta distância*] e [*terrestre, longa distância*].

Holland (HOLLAND, 1992) descreve o fenômeno onde a combinação de genes aumenta consideravelmente a performance de uma criatura, dado o seu ambiente. No exemplo anterior, podemos pensar em situações que favoreceriam um cromossomo ao invés de outro:

Cenário 1: Uma fase com diversas paredes estreitas, dificultando a visão do jogador e de inimigos. A fase é situada em um local com céu aberto.

Cenário 2: Uma fase que simula uma caverna, com paredes estreitas e muitas curvas. A caverna possui o teto muito baixo, dificultando a locomoção.

Nestes dois cenários, um cromossomo [*voador, longa distância*] garantiria uma vantagem considerável para o primeiro cenário, mas não para o segundo cenário, onde a possibilidade da criatura voar não é relevante. Caso o cenário sofresse alguma alteração durante o andamento do jogo, outros cromossomos poderiam apresentar uma vantagem em relação aos demais, podendo alterar o comportamento do inimigo.

Os cromossomos também podem reagir as ações do jogador, que por exemplo, pode optar por um estilo de jogo defensivo, onde busca esconder-se dos inimigos enquanto os ataca. Neste cenário, cromossomos que ofereçam algum método de ataque para oponentes defensivos estariam em vantagem, ocasionando em uma maior ocorrência deste cromossomo em próximas gerações. Esse comportamento é melhor descrito na seção 4.2.2.

4.1.2 *Fitness*

Após definirmos o ambiente, a população, a criatura e seus genes, ainda é necessário estabelecer uma métrica para o quão bem uma determinada criatura se saiu durante seu período de vida. Essa métrica, *fitness*, é intimamente atrelada ao tipo de problema a ser resolvido.

Uma criatura que precisa se movimentar na direção de um objetivo pode ter sua *fitness* atrelada a distância entre ela e o objetivo. Em outro caso, a *fitness* não é tão facilmente descoberta, sendo composta por múltiplos fatores.

A *fitness* também é responsável para direcionar a evolução e seleção de criaturas, que serão escolhidas com base neste valor. A seleção e utilização da *fitness* nesse contexto é detalhada posteriormente, em 4.2.2.

Utilizando o exemplo anterior, expandido em 4.1.1, podemos definir que a *fitness* será o tempo que o inimigo ficou vivo, fazendo com que aqueles inimigos que possuam mais defesas ou formas de esquivar dos ataques do jogador sejam considerados mais aptos. No entanto, e se todos os inimigos começarem a buscar por proteção, não atacando o jogador? Seriam considerados aptos, mas possivelmente o jogo deixaria de ser divertido. Neste caso, esse parâmetro não deve ser considerado, pelo menos unicamente, para determinar a aptidão de uma criatura.

Para a utilização de múltiplos parâmetros, podemos atribuir pesos a cada um deles. Dessa forma, podemos encontrar um balanço entre diversos atributos, com o peso de cada um ditando qual comportamento desejamos que a criatura obtenha. Assim, podemos ter uma combinação de tempo vivo, quantidade de ataques desferidos contra o jogador e ataques desviados, por exemplo.

Como citado anteriormente, descobrir quais parâmetros precisam ser considerados para a composição da *fitness* é extremamente importante, buscando evitar comportamentos indesejados.

4.2 ETAPAS

As etapas descritas nessa seção buscam orientar a criação de um algoritmo genético, utilizando os conceitos apresentados em 4.1. Além disso, uma relação com um jogo baseado em *waves* será traçada, usando dos exemplos de inimigos apresentados nas seções anteriores.

4.2.1 Inicialização

A etapa de inicialização é responsável por gerar a população inicial, garantindo uma variedade suficiente para tornar possível alcançar uma solução considerada aceitável. Múltiplos métodos que buscam reduzir custos computacionais e a eficiência de algoritmos genéticos podem ser aplicados (KAZIMIPOUR; LI; QIN, 2014), no entanto, a geração de maneira pseudoaleatória é a mais comum, devido a sua baixa complexidade de implementação.

Para a inicialização pseudoaleatória, *n* criaturas são geradas e seus genes são randomizados, garantindo criaturas variadas. No entanto, ao randomizar todas as criaturas, é possível gerar uma população inicial que não tem capacidade de alcançar a solução ideal.

Em um jogo baseado em *waves*, o processo de inicialização será a primeira *wave*, onde os inimigos ainda não conhecem seu ambiente e nem como as gerações anteriores performaram.

4.2.2 Seleção

A seleção é um passo essencial para o funcionamento de um algoritmo genético, buscando selecionar pais aptos para a criação de novas criaturas para a próxima geração, baseado em seus valores de *fitness*. Métodos para a seleção de pais variam de acordo com o propósito e o escopo da utilização dos algoritmos genéticos, podendo ser usados em conjunto ou isoladamente.

É possível a seleção de múltiplos pais, cada um contribuindo para a criação da nova criatura através dos processos descritos posteriormente na seção 4.2.3.

Da mesma forma como existem múltiplos métodos de seleção, também é possível aplicar diversas estratégias para selecionar pais aptos (LATA; YADAV; SOHAL, 2017). Iremos entrar em detalhes sobre duas dessas estratégias, comumente usadas em algoritmos genéticos: método de roleta e torneio.

Na utilização em um jogo com *waves*, a seleção pode ser realizada após o término de uma *wave*, coletando os dados de todas as criaturas que fazem parte da atual geração. Utilizando o ambiente apresentado na seção 4.1.1, podemos assumir que as criaturas com o cromossomo [*voador, longa distância*] seriam escolhidas caso o nível se passasse no cenário 1.

Para os métodos apresentados abaixo, consideraremos as seguintes criaturas, seguidas de seu número de *fitness*:

Tabela 1 – Criatura e sua *fitness*

Criatura	*Fitness*
1	0.5
2	0.3
3	0.8
4	0.1
5	0.9

4.2.2.1 Método de Roleta

Considerado um método tradicional e simples para a seleção de pais aptos, as criaturas são associadas a intervalos, ou partes de uma roleta virtual, proporcionais aos seus valores de *fitness*.

Para a seleção, um número pseudoaleatório é escolhido e a criatura que possui esse intervalo, ou pedaço da roleta é retornada como um pai. No entanto, caso um indivíduo possua uma *fitness* muito alta, este método pode acabar levando a convergência prematura (JEBARI, 2013).

Para melhor visualizar, consideraremos a seguinte roleta:

Podemos também separar a roleta em intervalos:

Depois de construída a roleta ou o intervalo, selecionamos um número aleatório entre [0, $\sum_{c=1}^{p} f_c$], onde c é uma criatura, p o número de criaturas na população e f a *fitness* associada com uma determinada criatura.

Nesse exemplo, precisamos escolher um número entre [0, 2.6]. Escolhido o número, encontramos em qual intervalo ele se encontra, selecionando assim a criatura retornada como pai. É preciso repetir esse processo n vezes, onde n é a quantidade de pais necessários para gerar uma nova criatura.

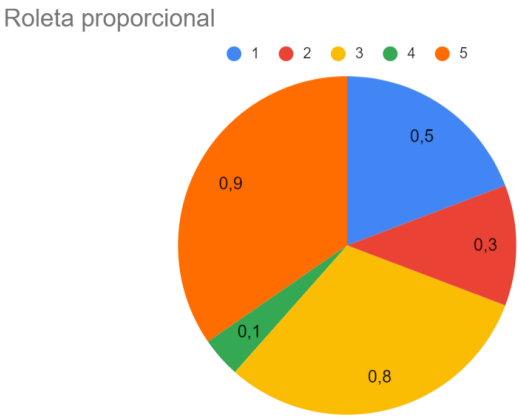


Gráfico 1 – Roleta criada a partir das criaturas apresentadas em 4.2.2.

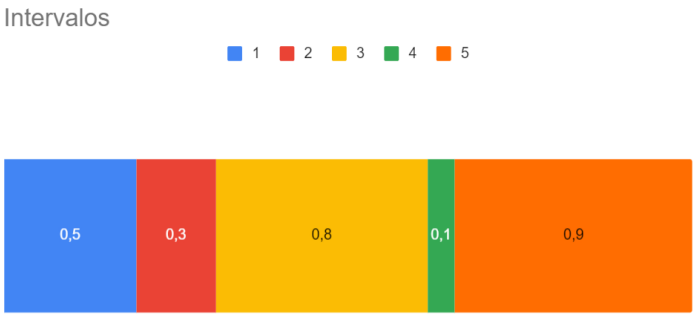


Gráfico 2 – Intervalo gerado.

Para a construção da nova geração, realizaremos o mesmo processo p vezes. É importante observar que, dado a natureza da construção da roleta, iterar por todos os membros da população a cada nova geração para a criação da roleta, além de iterar pelos membros da roleta até encontrar o intervalo necessário pode ser custoso em populações grandes.

4.2.2.2 Método de Torneio

No método de torneio, *k* criaturas são escolhidas, sendo a criatura escolhida aquela com maior *fitness* entre todos os participantes do torneio. Este método tende a ter taxas de convergência maiores de acordo com a quantidade de criaturas escolhidas para o torneio. O processo é repetido p vezes para a construção da nova geração.

Para Zhong et al (ZHONG et al., 2005), o método de torneio foi considerado superior ao roleta nos testes realizados, sendo mais eficiente em convergência do que o método anterior.

Para melhor visualizar, usaremos novamente as criaturas descritas na seção 4.1.1. Neste exemplo, consideraremos um torneio com tamanho 2 e portanto, selecionaremos 2 criaturas aleatoriamente da geração atual. Se considerarmos que para a criação de uma nova criatura 2 pais sejam necessários, precisaremos 2 torneios.

Quadro 2 – Torneios realizados.

Torneio	Criatura - *Fitness*	Vencedora
1	3 - 0.8 | 5 - 0.9	5
2	2 - 0.3 | 1 - 0.5	1

O mesmo processo é repetido 5 vezes, uma para cada novo membro da geração.

4.2.2.3 Elitismo

Durante gerações a população pode experimentar uma queda em sua *fitness*, onde criaturas com potencial sofreram mutações e seu desempenho caiu decorrente dessas mudanças.

Assim, durante a etapa de seleção, podemos criar clones dos melhores indivíduos, sem passar pelo processo de *crossover* ou de mutação.

Dessa forma, podemos nos assegurar que a qualidade de uma geração para a outra não sofrerá uma queda brusca.

4.2.3 *Crossover*

Após a seleção de pais aptos, é possível realizar o *crossover*, processo que busca combinar os cromossomos dos pais selecionados para geração de uma nova criatura. Métodos variados de *crossover* podem ser aplicados, com seus custos, vantagens e desvantagens dependendo do problema a ser resolvido ((UMBARKAR; SHETH, 2015). Dentre os diversos métodos, *crossover* com um ponto e *crossover* com dois ou k pontos são os mais utilizados.

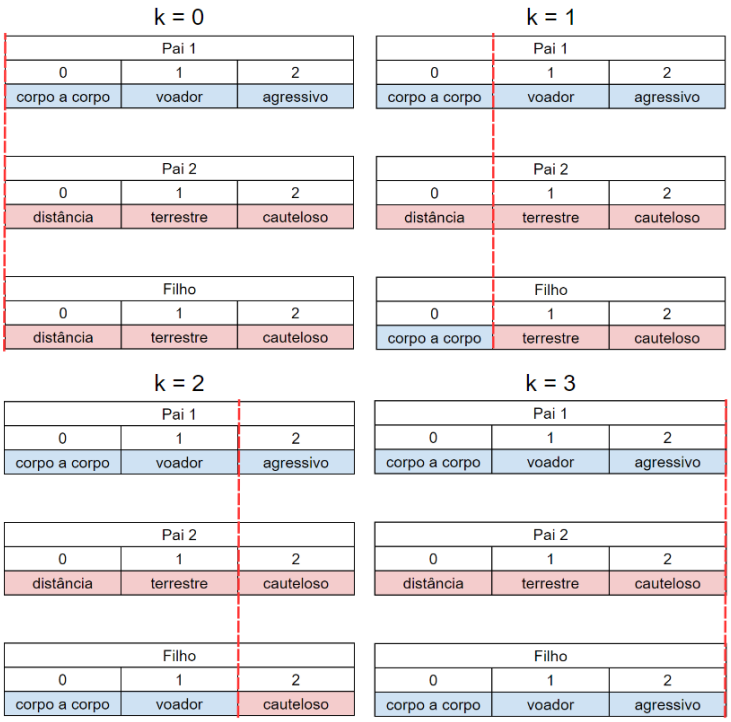
Para as próximas seções, consideraremos o seguinte cromossomo, representado por uma lista de genes: [*ataque*, *locomoção*, *comportamento*]. Além disso, apenas 2 pais são necessários para a geração de uma nova criatura.

É importante ressaltar que a implementação do cromossomo não é restrita a uma lista, da mesma forma que o método de *crossover* pode ser generalizado para n pais.

4.2.3.1 *Crossover* com um ponto

Neste tipo de *crossover*, um valor é selecionado entre $[0, c]$, sendo c o tamanho do cromossomo da criatura. Este valor será usado para determinar quais genes do primeiro pai serão utilizados para compor o cromossomos, com os restantes do segundo pai.

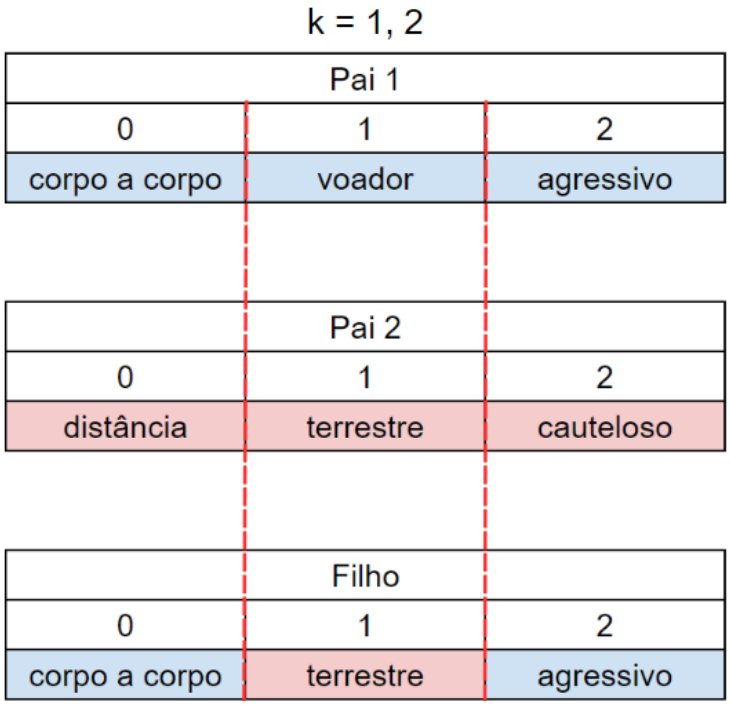
Figura 1 – *Crossover* com um ponto.



4.2.3.2 *Crossover* com 2 e k pontos

De maneira similar ao *crossover* com um ponto, dois valores são selecionados, separando os cromossomos dos pais em três pedaços. O cromossomo gerado para o filho é resultante dos cromossomos alternados entre pais, separados pelos dois pontos. O *crossover* com k pontos é uma generalização do método com dois pontos.

Figura 2 – *Crossover* com um dois pontos.



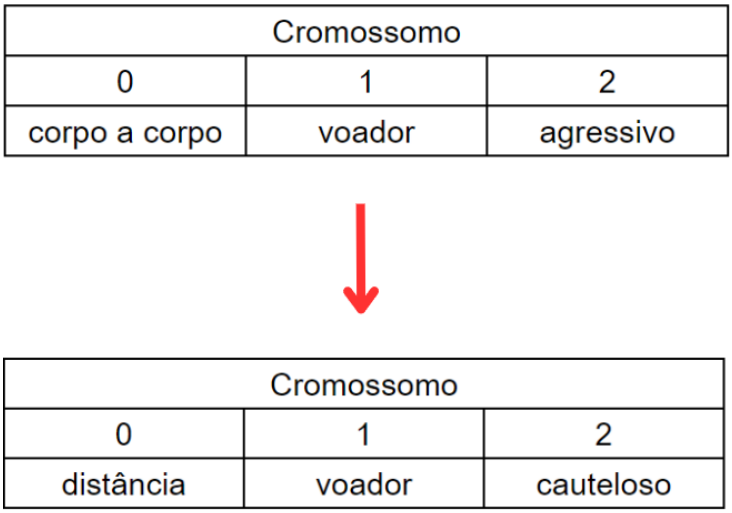
4.2.4 Mutação

O operador genético de mutação é utilizado para criar variedade após o *crossover*, buscando impedir uma convergência prematura, onde a população está presa em uma solução não aceitável, mas sem variedade para buscar outras alternativas (HASSANAT et al., 2019).

Dessa forma, podemos forçar mudanças nos genes das novas criaturas geradas após a seleção, introduzindo uma taxa de mutação. Assim, cada criatura tem uma chance de ter parcialmente ou totalmente seus genes randomizados.

Esse operador também torna possível a adaptação das criaturas de acordo com novos ambientes. Assim, mesmo que todas as criaturas possuam os mesmos cromossomos, ao alterar o ambiente e uma mutação ocorrer, possibilidades de soluções melhores podem surgir.

Figura 3 – Mutação realizada gene por gene.



4.2.5 Condição de parada

Um algoritmo genético precisa, em algum momento, parar de criar novas gerações. Essa condição de parada pode ser definida como um número máximo de gerações, quando não sabemos qual solução é considerada aceitável ou ela é muito complexa para ser alcançada, ou quando uma criatura chega em um certo nível de *fitness*, indicando que uma solução considerada aceitável foi encontrada.

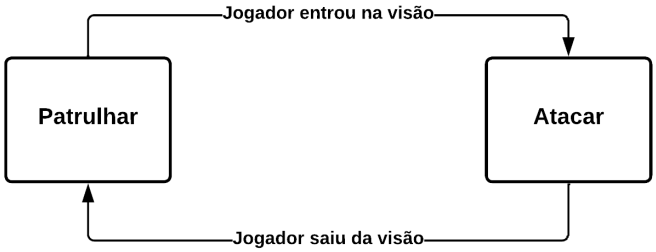
Em um jogo baseado em *waves*, a condição de parada pode estar relacionada com o fim do jogo, onde após n *waves* é considerado que o jogador venceu ou então com a morte do jogador, ocasionando em uma parada prematura.

5 CADEIA DE MARKOV E MÁQUINA DE ESTADOS

Agentes em jogos precisam tomar decisões de acordo com seu ambiente, como fugir ou atacar o jogador, auxiliar seus aliados, interagir com o cenário, entre outras. Para essas tomadas de decisões, a utilização de *finite-state machines* (*FSM*) é comum.

Uma máquina de estados pode representar apenas um estado em um determinado momento. Ao receber *inputs*, pode realizar a transição de um estado para o outro. Utilizando deste conceito simples é possível definir comportamento de agentes em um jogo, modificando seus estados de acordo com o ambiente e/ou o jogador. Na figura 4 é possível ver a representação visual de uma *FSM*, onde um agente possui dois estados: patrulhar e atacar. Inicialmente, o agente apenas deve patrulhar, ação que pode ser representada no jogo como andar em diversos pontos de uma área, por exemplo. Ao encontrar o jogador, o agente então deve buscar atacá-lo, voltando a patrulhar apenas quando o jogador sair de sua área de visão.

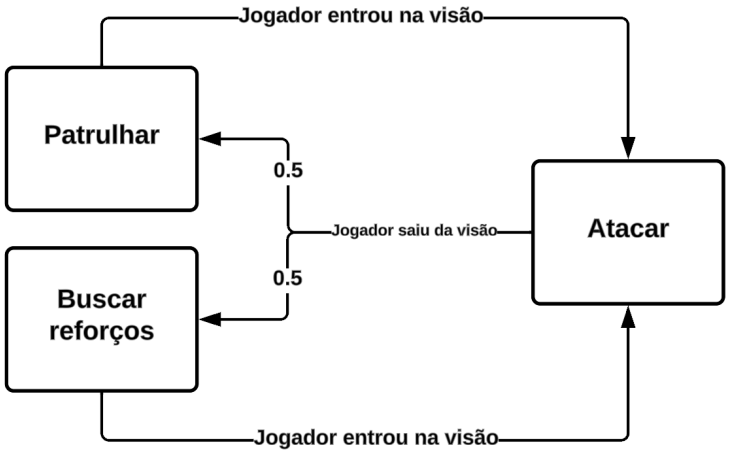
Figura 4 – Representação de uma máquina de estados.



No entanto, pela natureza determinística de uma máquina de estados, não é possível que agentes se comportem de maneira imprevisível. É esperado que dado as mesmas condições de entrada, o agente sempre tomará as mesmas decisões. Podemos aplicar os conceitos utilizados para a definição de uma cadeia de Markov, alterando a natureza determinística da máquina de estados e introduzindo um elemento de aleatoriedade para a tomada de decisões.

Utilizando uma estrutura semelhante a ilustrada na figura 4, podemos entender as transições realizadas de maneira probabilística. Ao perder de vista o jogador, ao invés de voltar a patrulhar, o agente pode decidir buscar reforços. Essas tomadas de decisões, ilustradas na figura 5, possuem probabilidades de serem efetuadas. É importante ressaltar que os valores utilizados neste capítulo são meramente ilustrativos, já que a atribuição de probabilidades em um jogo real é um processo realizado empiricamente através de testes.

Figura 5 – Representação de uma cadeia de Markov implementada como uma máquina de estados.



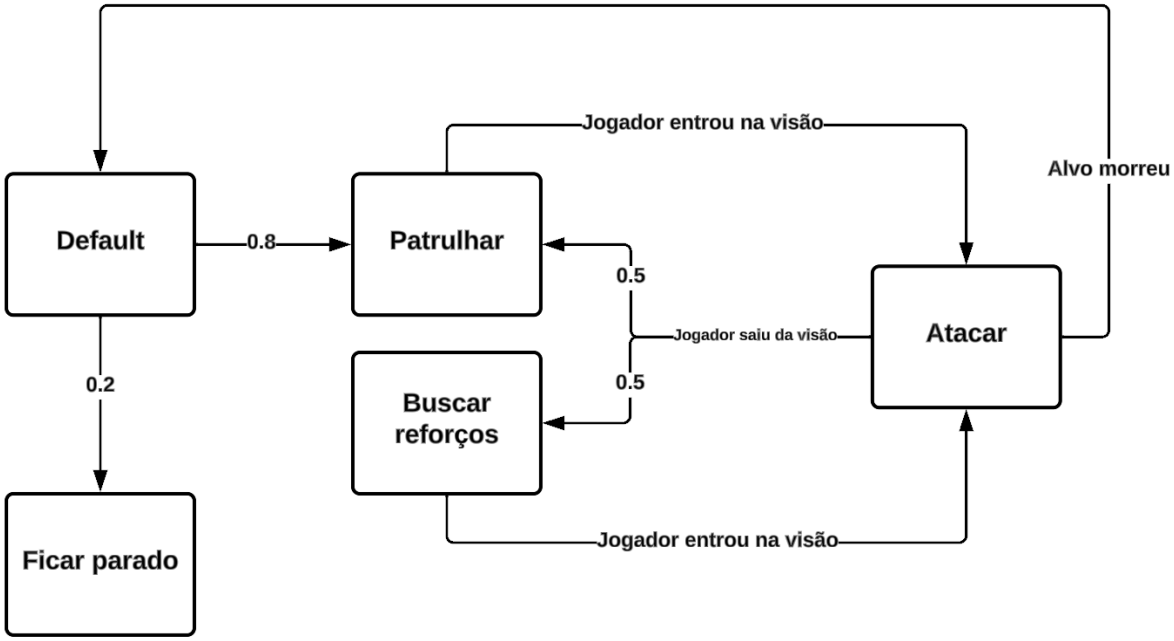
Da mesma forma como a máquina de estados representada anteriormente, essa implementação também requer que apenas um estado esteja ativo em um determinado momento. Ao chegar em uma condição, um dos estados será escolhido, com base na probabilidade de sua ocorrência.

Um estado não depende de um estado ou conhecimento anterior para a sua tomada de decisão, mantendo a propriedade markoviana da estrutura.

Servindo como um ponto de entrada em nossa máquina de estados, podemos utilizar um estado *default*, que dita as primeiras transições da máquina. Definimos também que, caso a soma das probabilidades das transições seja menor do que 1, o valor faltante indica uma transição para o estado *default*. Assim, possuímos uma forma de simplificar as transições, mantendo um ponto de saída para estados que queiram reinicializar a máquina de estados.

Para transições a apenas um estado e que não apresentam uma indicação de probabilidade, assumimos que a probabilidade é 1. Além disso, as transições que possuem uma das probabilidades sendo o próprio estado não podem ser testadas duas vezes. Assim, não é necessário representar visualmente múltiplos estados diferentes com transições bloqueadas. As convenções definidas neste capítulo são necessárias para a simplificação das máquinas de estado representadas no capítulo 6.

Figura 6 – Implementação de uma máquina de estados com um estado *default*.



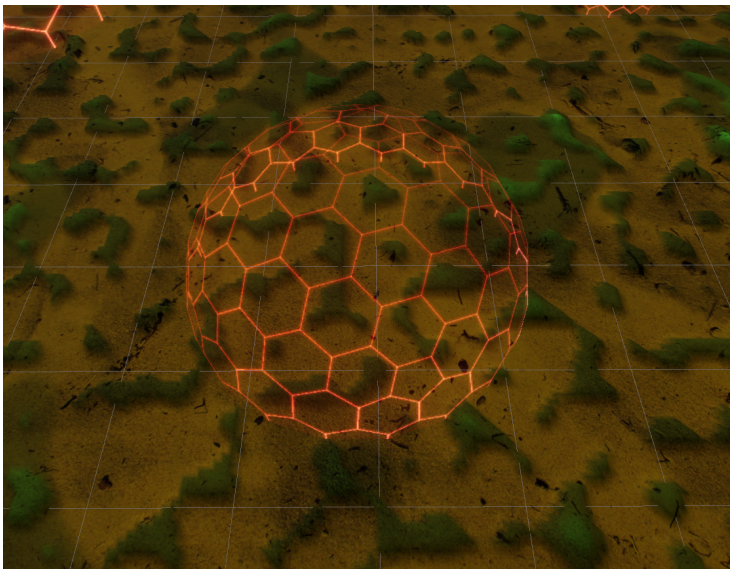
6 DESENVOLVIMENTO

Para a avaliação do desempenho da geração procedural, um jogo 3D baseado em *waves* foi desenvolvido.

A cada partida, dois grupos de agentes competem entre si. O objetivo final é da destruição do outro grupo, utilizando de armas e outros artifícios para alcançar este objetivo.

No começo de cada partida, um mapa simples é gerado de maneira procedural, utilizando de *perlin noise* para criar pequenos relevos. Além disso, a arena onde a partida ocorre é delimitada por um domo, impedindo que os agentes pertencentes a cada grupo saiam da área especificada.

Figura 7 – Visão aérea do terreno gerado, juntamente com o domo da arena.



Uma partida é constituída de diversas rodadas ou *waves*, definidas por um período de tempo. Após esse período, a *wave* é considerada como finalizada e os procedimentos para a criação de uma nova rodada são iniciados.

Em cada nova rodada, uma geração é criada utilizando da seleção através do método de roleta, melhor descrito na seção 4.2.2.1 e os operadores genéticos descritos no capítulo 4. Os fatores levados em consideração para a construção da *fitness*, bem como os genes que compõem a criatura estão detalhados na seção 6.2.2.

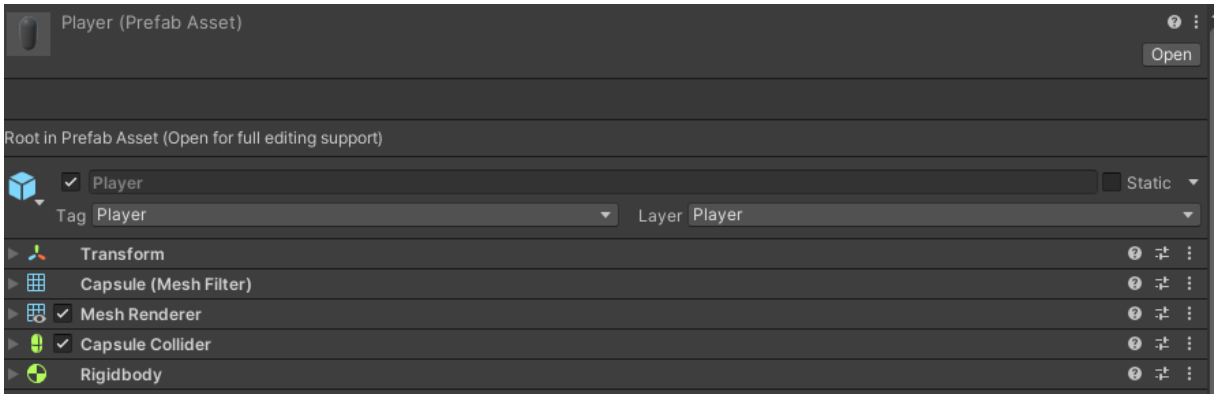
Pela natureza criativa envolvida no processo de desenvolvimento de um jogo, a escolha dos tipos de armas, *traits* e comportamentos foram decididas através da inspiração de outros jogos semelhantes, com os valores para diferentes atributos tendo sido ajustados de maneira empírica após a realização de testes.

6.1 ARQUITETURA

Para melhor acomodar o uso de GA, uma arquitetura flexível foi necessária, onde diferentes comportamentos e funcionalidades poderiam ser associadas a um objeto, sem uma referência direta a outras funcionalidades acopladas.

Em *Unity*, a *engine* escolhida para a implementação do projeto, um componente é qualquer classe acoplada em um determinado *GameObject* - representação de um objeto com múltiplos componentes dentro do jogo. Componentes podem realizar múltiplas funções, como controlar a movimentação do objeto, manejar seus ataques, determinar sua interação com a física do mundo, entre outras.

Figura 8 – Exemplo de componentes de um *GameObject*



Os genes de uma criatura possuem a capacidade de adicionar, remover ou alterar componentes. Portanto, um sistema onde um componente não depende diretamente de outro é extremamente útil. Dessa forma, podemos adicionar armas diferentes, formas de movimentação, diferentes elementos visuais ou qualquer outra funcionalidade sem a dependência de uma outra já existir.

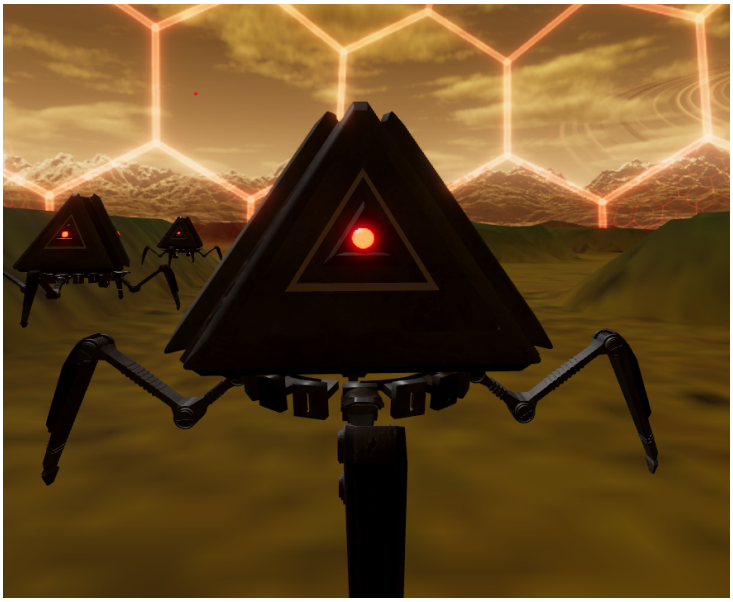
Mesmo assim, a comunicação entre componentes ainda é desejável, mesmo que não estritamente necessária. Para isso, um sistema de eventos foi desenvolvido, permitindo *GameObjects* escutarem e dispararem diferentes eventos.

6.2 AGENTES

Todos os inimigos presentes no jogo são gerados de maneira procedural, buscando se adaptar ao estilo de jogo do jogador através de dados coletados ao longo de uma rodada. Ao longo das próximas seções, poderemos entender as principais características e os genes que compõem o agente.

Os termos agente, inimigo e criatura serão usados como sinônimos ao longo do capítulo.

Figura 9 – Representação visual do agente.



6.2.1 Atributos

Semelhante ao trabalho de Olsson (OLSSON, 2019), atributos como vida e velocidade de ataque foram implementados para os agentes. Cada atributo possui um valor inicial, um valor mínimo e um valor máximo. A seguir, os atributos presentes em todos os agentes:

Quadro 3 – Atributos.

Atributo	Valor inicial	Valor mínimo	Valor máximo
Vida	5	0	30
Velocidade de movimento	4	0.5	12
Tempo de invulnerabilidade	0.1	0.1	1
Multiplicador de dano comum	0	0	1
Multiplicador de dano explosivo	0	0	1
Multiplicador de cura	0	0	1
Multiplicador de taxa de tiro	0	0	1
Redução de dano	0	0	1
Quantidade de projéteis	0	0	1

Atributos como vida e velocidade de movimento são utilizados para as funções básicas do agente, como andar e morrer. Multiplicadores são aplicados em eventos em que o agente teve participação. Além disso, multiplicadores podem assumir valores negativos. Assim, uma criatura com -0.5 em redução de dano receberá 50% a mais de dano ao ser atingida.

6.2.2 *Fitness*

Como descrito em maior detalhes na seção 6.2.2, a *fitness* do agente é fator fundamental em determinar seu comportamento. Ao longo da rodada, através do sistema de eventos, estatísticas referentes a criatura são coletadas para serem usadas na construção de sua *fitness*. Essas estatísticas coletadas, em conjunto com seus pesos, constituem uma lista de propriedades. Uma propriedade, além de possuir uma estatística e um peso relacionados, também possuem um valor bruto e um valor final. O valor bruto é simplesmente o valor coletado pela estatística, sem nenhum tipo de alteração.

Para a obtenção do valor final v_f, o valor máximo v_m encontrado em todas as criaturas da mesma geração referente a mesma estatística é utilizado, junto de um peso p que representa o quão importante a propriedade é para a construção da *fitness*. O peso precisa estar entre 0 e 1, sendo que $1 = \sum p$.

$$v_f = p * (v_b / v_m)$$

Uma propriedade também pode ser indicada como inversa, onde o quanto mais alto seu valor bruto, menor seu valor final.

$$v_f = p * (1 - (v_b / v_m))$$

Em ambos os casos, o valor final está sempre entre 0 e 1.

Após o cálculo de todas as propriedades, a *fitness* é encontrada através da soma de todos os valores finais.

Quadro 4 – Propriedades.

Propriedade	Inversa	Descrição
DamageHitsDealt	Não	Quantidade de ataques desferidos
DamageDealt	Não	Dano infligido
DamageHitsTaken	Sim	Quantidade de ataques recebidos
DamageTaken	Sim	Dano recebido
HealingHitsDealt	Não	Quantidade de curas desferidas
HealingDealt	Não	Cura recebida
HealingHitsTaken	Não	Quantidade de curas recebidas
HealingTaken	Não	Cura recebida
FriendlyFireDealt	Sim	Dano infligido contra aliados
FriendlyFireTaken	Sim	Dano recebido de aliados
Alive	Não	Ficou vivo até o fim da rodada

6.2.3 Genes

Os genes ditam as ações de um inimigo, determinando que tipo de arma, *traits* e comportamento possui.

6.2.3.1 Armas

O gene é composto apenas do tipo de arma que será utilizado pelo inimigo, com os dados da arma definidos de maneira global para todos os inimigos. Ao sofrer mutação, o tipo de arma será escolhido de maneira randômica entre todos os tipos possíveis de armas.

Para uma maior diversidade, 6 tipos de armas foram desenvolvidas, sendo que uma criatura pode possuir apenas uma arma.

Todas as armas, possuem o atributo *cooldown* e, com exceção do escudo, um identificador do tipo de ataque causado pela arma.

O atributo da arma *cooldown* c em conjunto com o atributo do agente *multiplicador de taxa de tiros* mtx dita a taxa de tiros tx de uma arma - ou o intervalo de tempo entre utilizações da arma.

$$tx = c - (c * mtx)$$

O tipo de ataque por sua vez pode ser classificado de três formas: comum, cura e explosivo. É possível obter o dano ou cura realizado pela arma através de seu dano ou cura base em conjunto com os multiplicadores do agente para dano comum, explosivo ou cura. Assim, podemos definir que, para um dano ou cura base b e o multiplicador m correspondente do agente, o valor final f de cura ou dano é o seguinte:

$$f = b + (b * m)$$

Cada arma é descrita em maior detalhe a seguir, junto de seus valores iniciais. Valores iniciais foram ajustados empiricamente, após testes e tentativas para obter um balanceamento mínimo. No entanto, como é discutido no capítulo 7, criaturas possuíram uma preferência por apenas duas armas, indicando que um desbalanceamento ainda ocorreu. Além disso, em armas como granadas e bombas, o fogo amigo não foi implementado já que, após testes, essas armas não funcionavam de maneira correta pela natureza dos comportamentos implementados, que não buscam evitar ataques em área.

6.2.3.1.1 Escudo

Ao ser usada, um escudo é criado na direção do alvo da criatura. É capaz de bloquear tiros e quebra após um determinado tempo ou até sua vida acabar.

Quadro 5 – Escudo.

Atributo	Valor
Cooldown	10s
Segundos até quebrar	8
Vida	5

6.2.3.1.2 Tiros rápidos de dano

A criatura atira um projétil rápido na direção de seu alvo, causando dano. Também pode a vir a causar dano em aliados. Ao acertar um aliado, o valor de dano é multiplicado pelo atributo *fogo amigo*.

Quadro 6 – Tiros rápidos de dano.

Atributo	Valor
Cooldown	0.2s
Tipo de dano	Comum
Dano	0.4
Fogo amigo	0.2

6.2.3.1.3 Tiros guiados de dano

A criatura cria projéteis que avançam na direção de um alvo inimigo próximo, causando dano. Não acerta aliados e tem capacidade de perseguir seu alvo.

Quadro 7 – Tiros guiados de dano.

Atributo	Valor
Cooldown	0.8s
Tipo de dano	Comum
Dano	0.25
Projéteis	4

6.2.3.1.4 Tiros guiados de cura

A criatura cria um projétil que avança na direção de um alvo aliado próximo, curando-o. Não é capaz de curar inimigos e tem capacidade de perseguir seu alvo.

Quadro 8 – Tiros guiados de cura.

Atributo	Valor
Cooldown	1s
Tipo de dano	Cura
Cura	0.15
Projéteis	5

6.2.3.1.5 Granadas

A criatura lança um projétil na direção de um alvo inimigo que após um tempo causa uma explosão em área, causando dano em inimigos.

Quadro 9 – Granada.

Atributo	Valor
Cooldown	2.5s
Tipo de dano	Explosivo
Dano	1
Tempo para explosão	1.5s
Raio de explosão	4u

6.2.3.1.6 Bomba

Um ponto centrado em um inimigo próximo é definido, causando uma explosão em área na posição do ponto após passado um determinado tempo. Causa dano apenas em inimigos.

Quadro 10 – Bomba.

Atributo	Valor
Cooldown	7s
Tipo de dano	Explosivo
Dano	3
Tempo para explosão	4s
Raio de explosão	8u

6.2.3.2 *Traits*

A presença de habilidades, *power-ups*, modificações em atributos ou itens é um conceito comum em jogos do gênero *roguelike*. Da mesma forma, um sistema foi criado para suportar a adição de *traits* - características especiais que buscam alterar os atributos do inimigo.

O gene de *traits* é responsável por manter uma lista do tipo de *traits* que o inimigo possui. Ao sofrer mutação, um *trait* é removido da lista e outro é adicionado.

Para comportar a adição de múltiplos *traits* com facilidade e buscando integração com GAs sem um sistema rígido, todo agente possui um componente *TraitController*, que é encarregado de identificar possíveis *traits* e controlar todo processo de adição e remoção. Inicialmente, a primeira geração de criaturas recebe um *trait* da lista de *traits* disponíveis.

Para criar comportamentos mais interessantes, após n rodadas a criatura pode escolher um novo *trait* para adicionar ao seu gene. Essa escolha é baseada em um fator de atratividade, valor entre 0 e 1 associado ao *trait* que determina o quão bem criaturas que o utilizaram desempenharam em rodadas anteriores. Por padrão, todos os *traits* possuem um fator de atratividade de 0.5.

Para realizar essa alteração, a *fitness* de uma criatura precisa estar dentro de um intervalo para modificar o fator de atratividade negativa e positivamente. Para a modificação, a criatura utiliza sua *fitness* multiplicada por um valor *x*, onde *x* é uma constante e serve para diminuir ou aumentar a velocidade de mudança dos fatores de atratividade.

O processo de atualização do fator de atratividade é realizado a cada *wave* com todas as criaturas da geração. Ao começar a escolha por um novo *trait*, a criatura recebe *k* opções únicas de *traits* e, de maneira semelhante ao método de seleção por roleta, apresentado na seção 4.2.2.1, escolhe um novo *trait*.

Para evitar a dominância de um determinado *trait* com fator de atratividade muito alto, cada criatura possuí uma taxa de ignorância, que determina o quão propensa a criatura está de tomar uma decisão sem levar em consideração os fatores de atratividade e apenas escolher de maneira randômica dentre as opções oferecidas.

Além disso, uma criatura pode possuir um número finito do mesmo *trait*, definido na criação do *trait*. Assim, o acúmulo excessivo de uma determinada característica não é permitido.

6.2.3.3 Comportamento

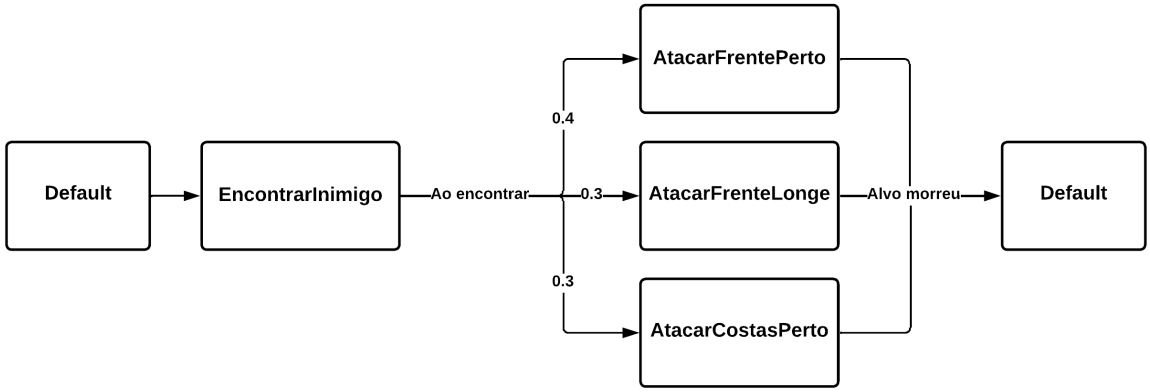
O gene de comportamento é responsável por armazenar um tipo de comportamento, utilizado para determinar quais ações o inimigo será capaz - suporte, tático, defensivo e agressivo. De maneira similar aos outros genes, ao sofrer uma mutação o tipo de comportamento da criatura é randomizado, podendo assumir qualquer um dos quatro valores citados anteriormente.

Para a tomada de decisões, a implementação de uma cadeia de Markov, representada por uma máquina de estados foi realizada. A implementação é descrita com detalhes no capítulo 5. Todos os comportamentos operam apenas com combinações de dois estados: *EncontrarAlvo* e *MoverBaseadoAlvo*. As probabilidades utilizadas para cada transição foram definidas empiricamente, através de testes para obter valores que tornassem os comportamentos mais interessantes.

6.2.3.3.1 Agressivo

O comportamento agressivo representa um agente simples, que busca inimigos e escudos inimigos para atacar, não levando em consideração posição de aliados.

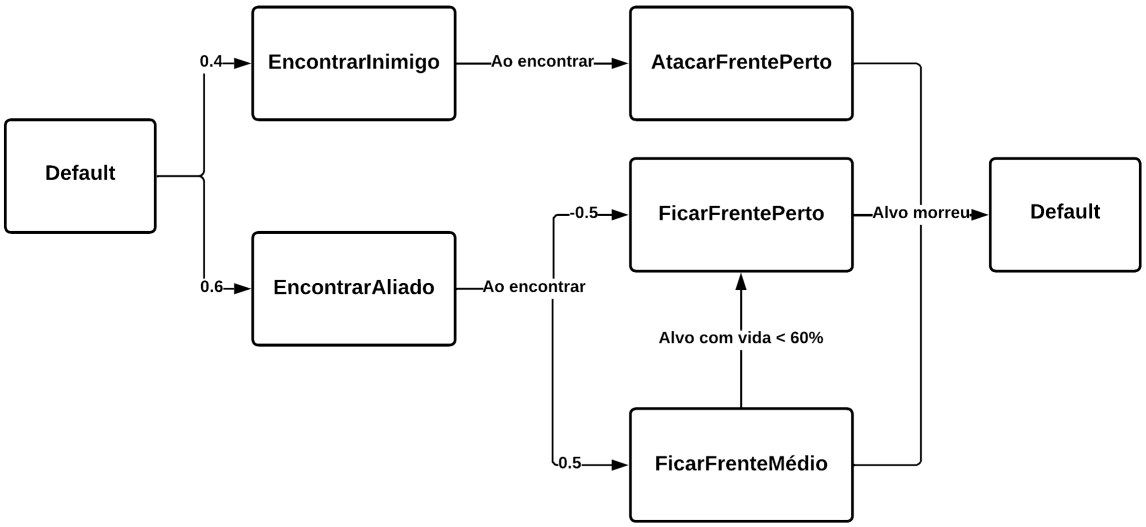
Figura 10 – Comportamento agressivo.



6.2.3.3.2 Defensivo

O comportamento defensivo representa um agente simples, que busca interceptar ataques inimigos ao se manter na frente de aliados ou então ao se manter perto de inimigos.

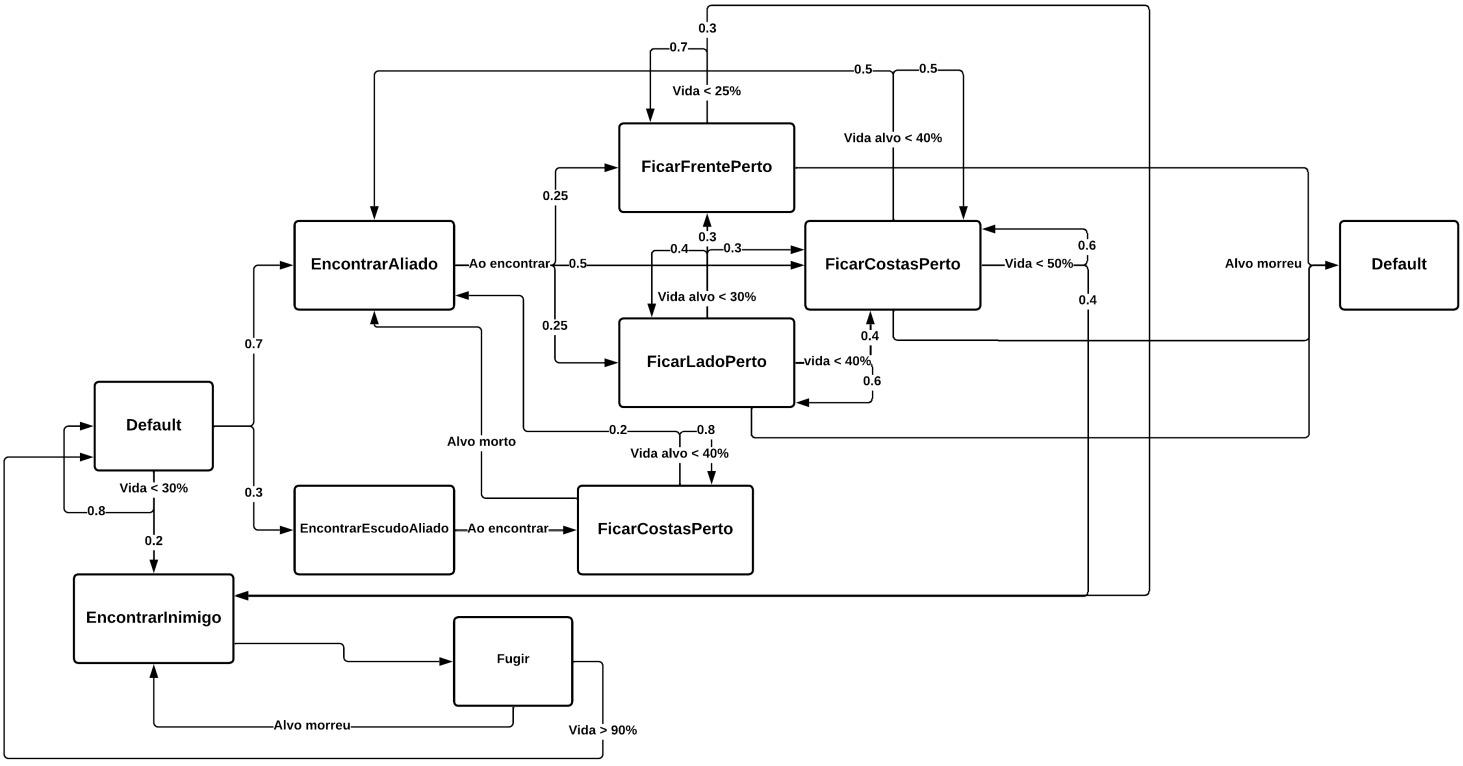
Figura 11 – Comportamento defensivo.



6.2.3.3.3 Suporte

O comportamento suporte representa um agente complexo, que busca escudos e criaturas aliadas para se proteger. Possui uma leve tendência a proteger aliados, ao invés de buscar sua própria sobrevivência. Também toma diversas decisões baseadas na quantidade de vida própria e de seu alvo.

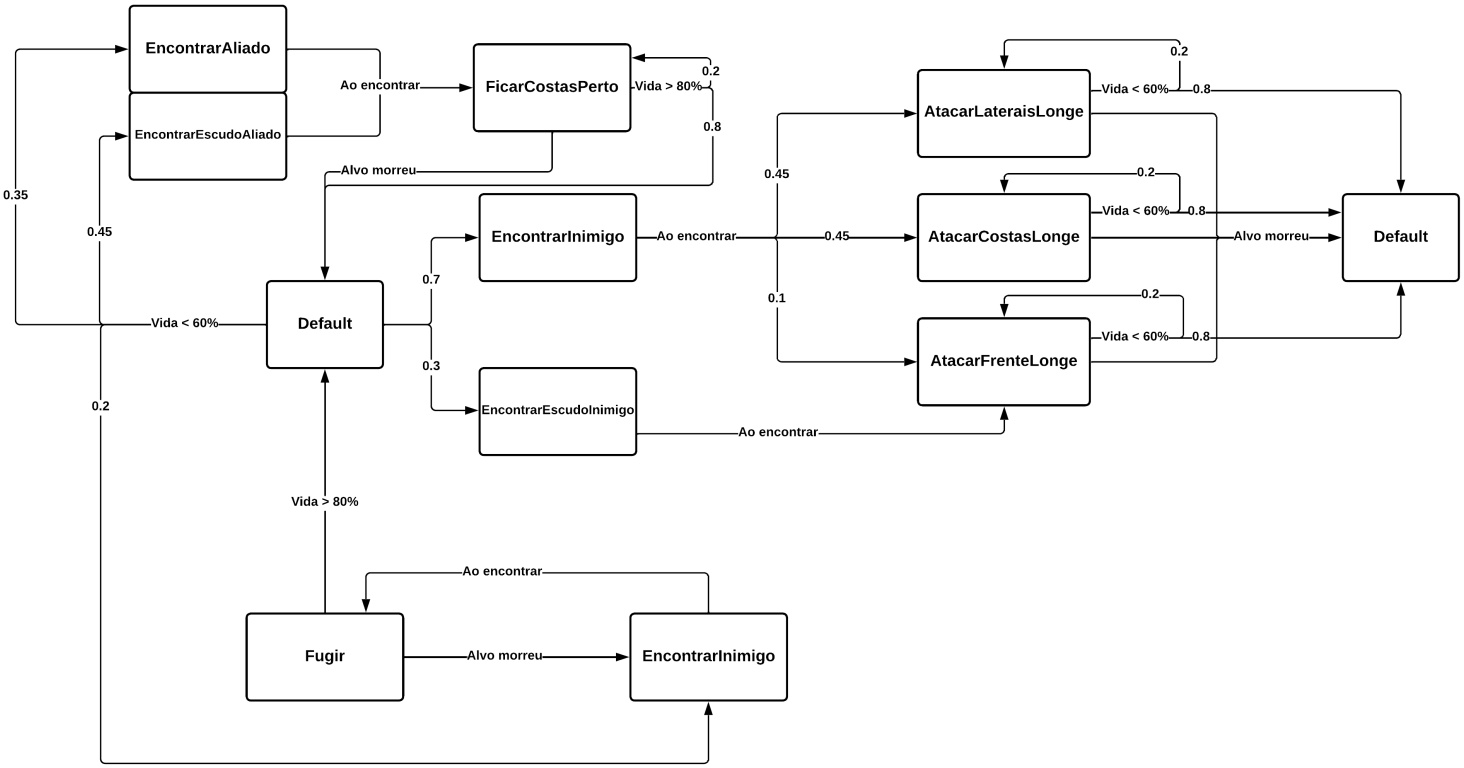
Figura 12 – Comportamento suporte.



6.2.3.3.4 Tático

O comportamento tático representa um agente complexo, que busca destruir escudos e criaturas inimigas, além de buscar a sua própria preservação ao fugir ou encontrar aliados para se esconder. Diferentemente do comportamento agressivo, criaturas táticas tem preferência por ataques laterais ou nas costas de inimigos.

Figura 13 – Comportamento tático.



6.3 FERRAMENTAS UTILIZADAS

O jogo implementado para o uso de algoritmos genéticos foi desenvolvido na linguagem C#, usando a engine *engine* Unity, versão 2021.3.11f1, devido a sua grande popularidade e também familiaridade do autor. A modelagem dos elementos 3D foi realizada por meio do software Blender e o versionamento foi feito através do GitHub.

7 AVALIAÇÃO

Para avaliar a capacidade de adaptação dos agentes gerados através de algoritmos genéticos, três grupos de agentes foram colocados para batalhar. Buscou-se observar a mudança de armas, comportamentos e *traits*, de acordo com seus oponentes e com os pesos utilizados para a construção da *fitness* da população. Todos os grupos possuem os mesmos valores iniciais para atributos, armas e os mesmos *traits* e comportamentos disponíveis, descritos anteriormente no capítulo 6.2.

Os três grupos são gerados inicialmente de maneira aleatória. O grupo 1 não sofre alterações, servindo essencialmente como uma ameaça imutável. Para a inicialização, a mesma *seed* foi usada para todos os testes com o grupo 3 para melhor visualizarmos as diferenças entre métodos de seleção e propriedades de *fitness*. O grupo 2, por sua vez, é randomizado a cada *wave*, atuando como uma ameaça imprevisível e que não é capaz de se adaptar. O grupo 3 utiliza de GA para geração dos agentes em cada *wave*. Ao longo dos embates, os pesos das propriedades para a obtenção da *fitness* é alterado entre três tipos: balanceado, ofensivo e defensivo. O valor das propriedades pode ser visto nos gráficos abaixo.

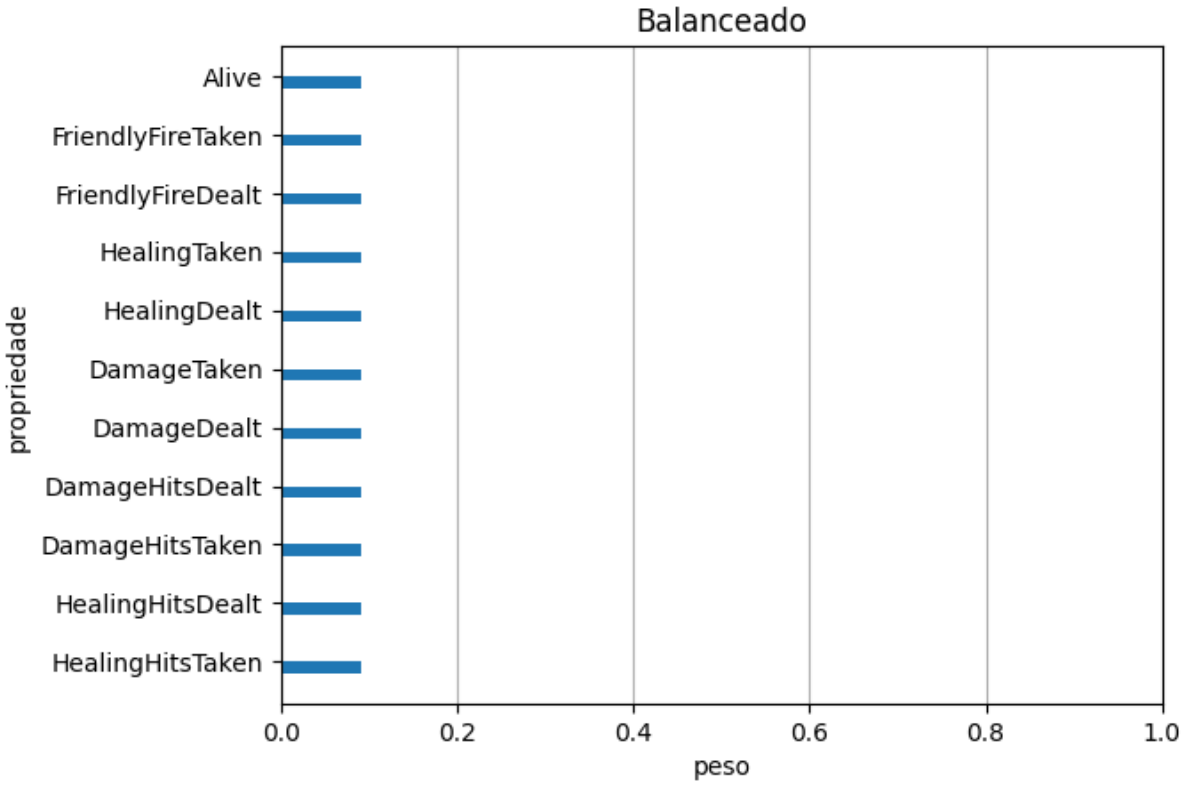


Gráfico 3 – Balanceado.

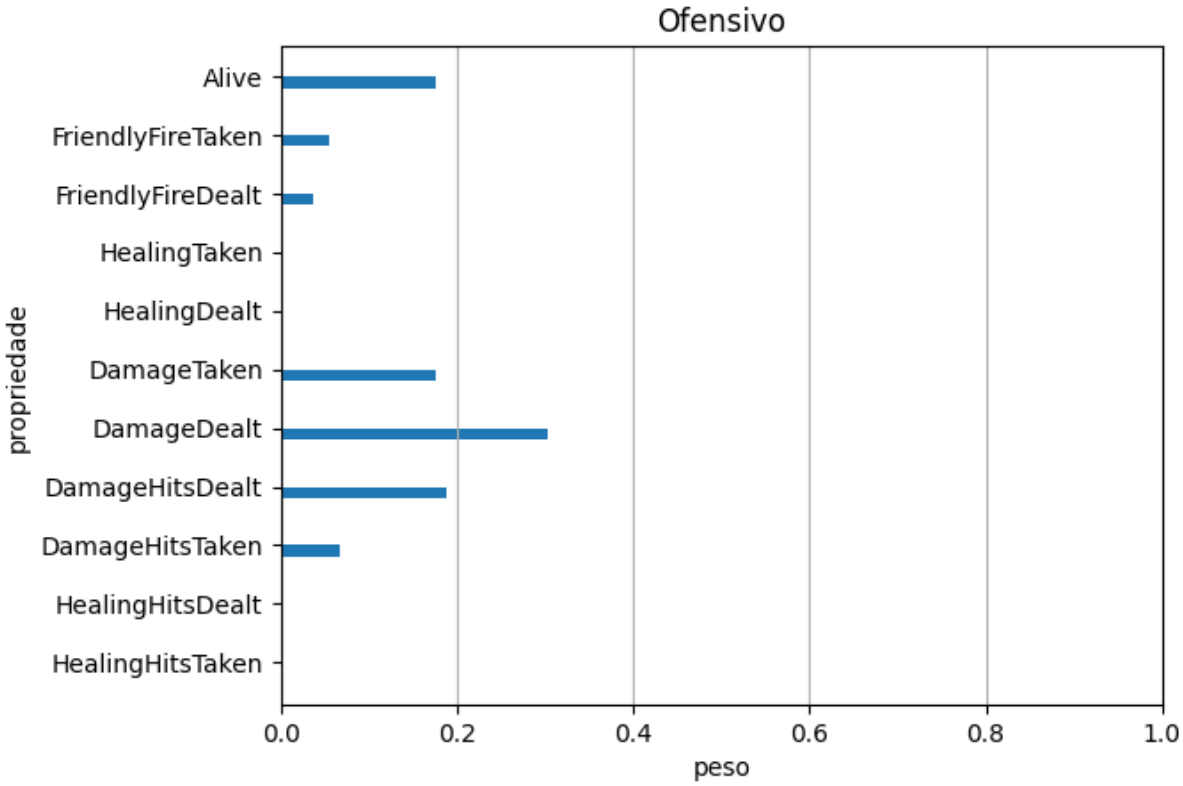


Gráfico 4 – Ofensivo.

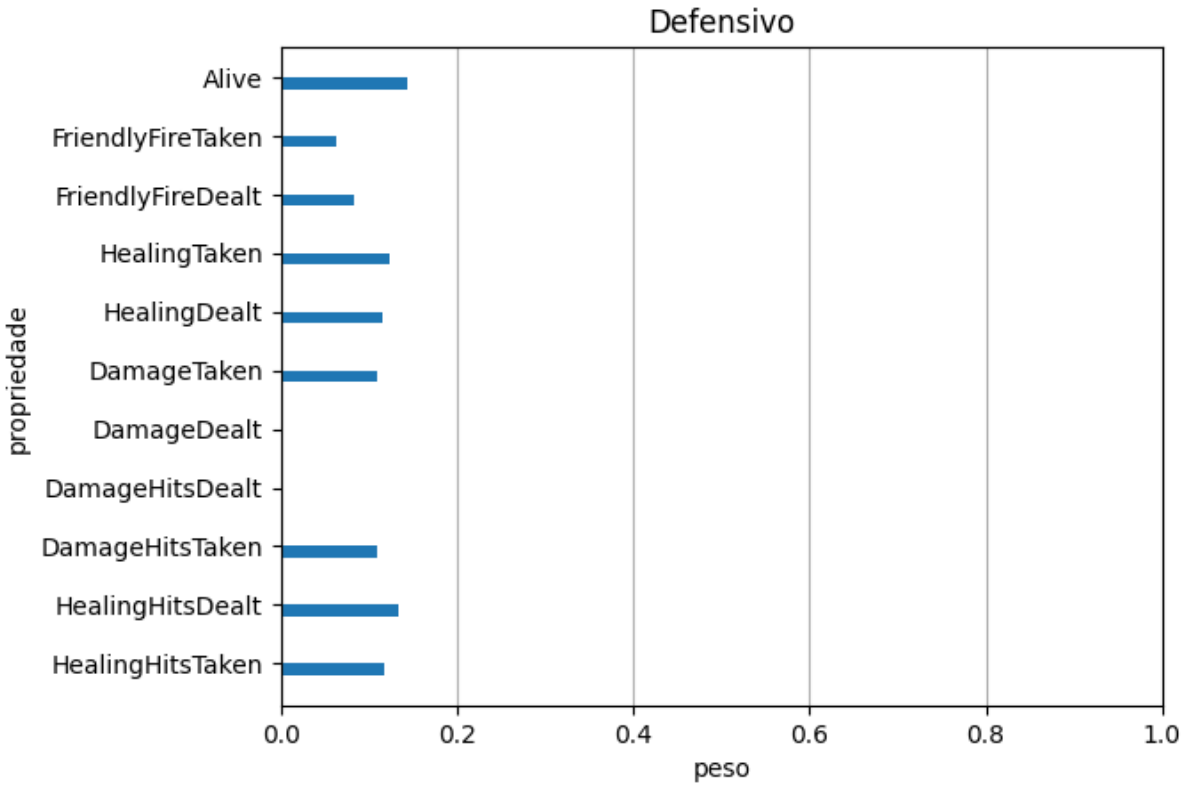


Gráfico 5 – Defensivo.

Todos os embates utilizam 20 agentes em cada grupo, totalizando 40 agentes em uma partida. Foram realizadas 20 *waves* para cada embate, cada uma durando 60 segundos. Além disso, para os grupos 2 e 3, é possível obter 6 *traits* ao longo da partida, com um novo *trait* sendo oferecido a cada 2 *waves*. Para o grupo 3, os seguintes valores foram usados nos embates:

Quadro 11 – Valores utilizados para a geração de criaturas com GA.

Propriedade	Valor
Taxa de mutação	0.2
Quantidade de pais	4
Opções de *traits*	4
Taxa de ignorância	0.15
fitness para diminuição da taxa de atratividade	0.3
fitness para aumento da taxa de atratividade	0.7
Peso para alteração da taxa de atratividade	0.02
Elitismo	5
Tamanho do torneio	4

Para avaliar o desempenho do grupo 3, que foi gerado proceduralmente, métricas como a quantidade de criaturas mortas e a *fitness* mínima e máxima foram utilizadas, buscando entender de maneira mais ampla o impacto do uso da geração procedural na partida. Além disso, a quantidade de *traits*, armas e comportamentos utilizados por geração também foi analisada, procurando compreender a relação entre os genes da criatura com o seu desempenho na partida.

7.1 GRUPO 1 X GRUPO 3

Primeiramente, podemos ter uma visão geral dos resultados obtidos nesse embate através dos gráficos 6, 7 e 8.

É possível visualizar que, quando utilizados o estilo de jogo ofensivo e defensivo, onde para estatísticas específicas pesos muito altos foram atribuídos, temos uma diferença significativa na criatura com maior *fitness* para a menor. O balanceado, por possuir diversos pesos e abranger outros estilos de jogo, possui um mínimo e máximo muito mais contidos. Tanto na média de *fitness* quanto no mínimo e máximo, os métodos de torneio e roleta não apresentam grandes diferenças.

Na quantidade de criaturas mortas por geração, é possível observar uma diferença entre os métodos de seleção quando temos estilos de jogo balanceado e defensivos. Podemos observar que, nos casos onde o estilo balanceado foi usado, o acréscimo em criaturas mortas ao fim da partida do grupo 1 quando o método de roleta foi usado coincide com um acréscimo na utilização de criaturas com bombas.

Da mesma forma, o período onde criaturas do grupo 1 possuem as maiores mortes também coincide com uma maior utilização de bombas, quando observamos o método de torneio. Para o estilo defensivo, não é possível notar uma indicação de combinação entre *traits*, armas e comportamentos que justifiquem a diferença.

Gráfico 6 – Grupo 3, GA: *Fitness* médio das criaturas geradas proceduralmente. (Versus Grupo 1, imutável)

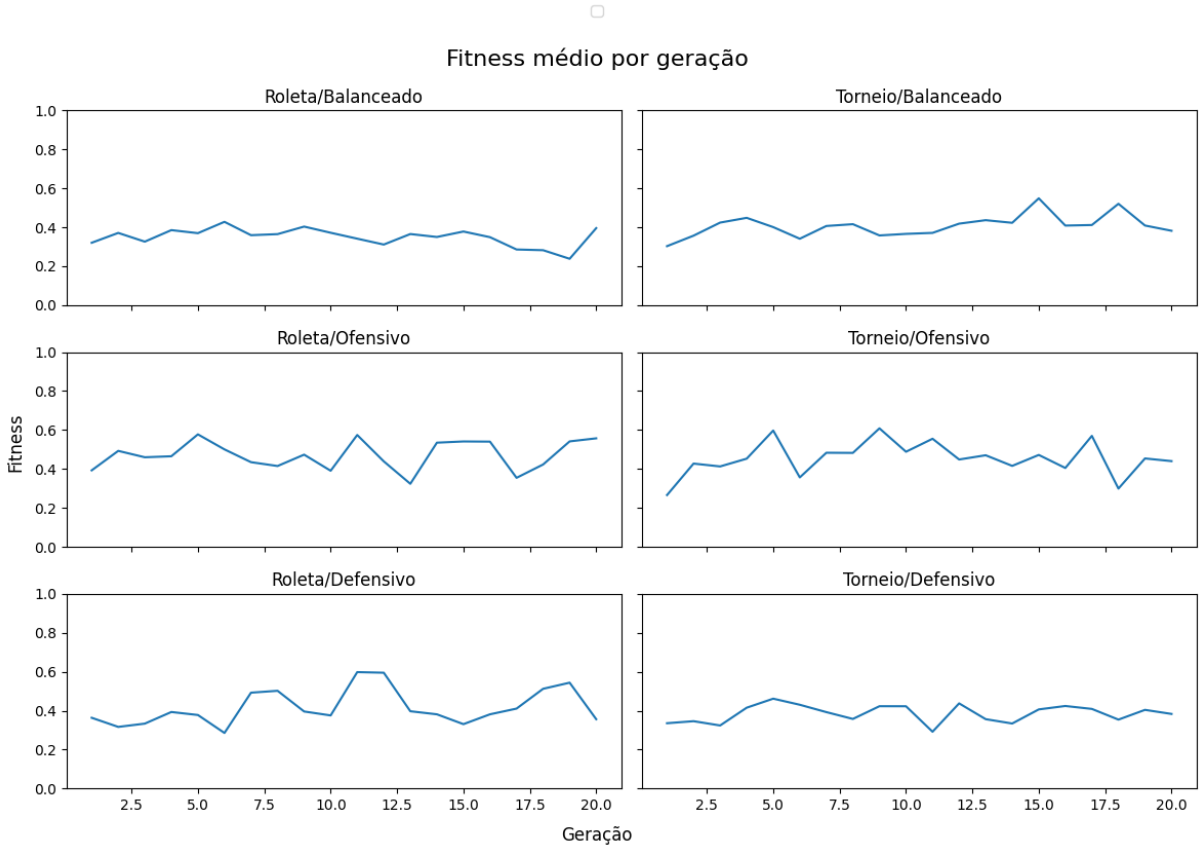


Gráfico 7 – Grupo 3, GA: *Fitness* mínimo e máximo das criaturas geradas proceduralmente. (Versus Grupo 1, imutável

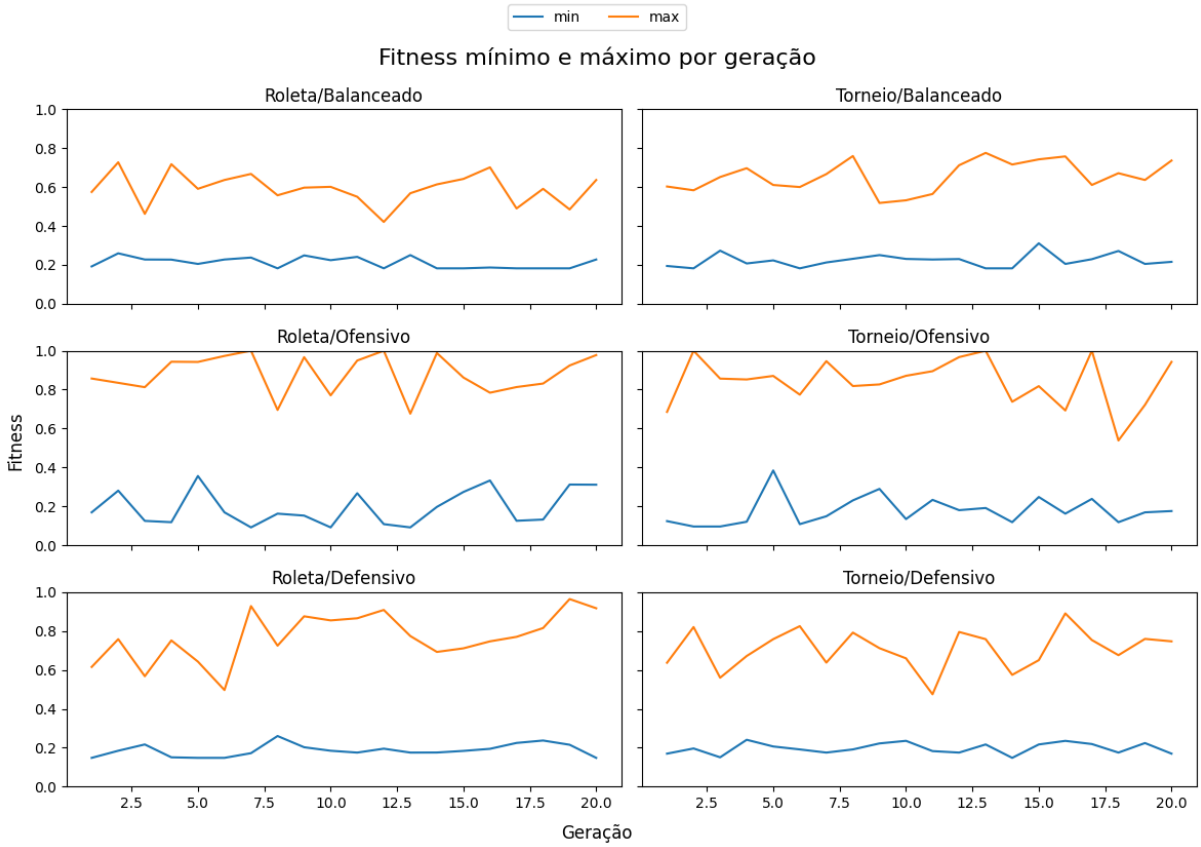
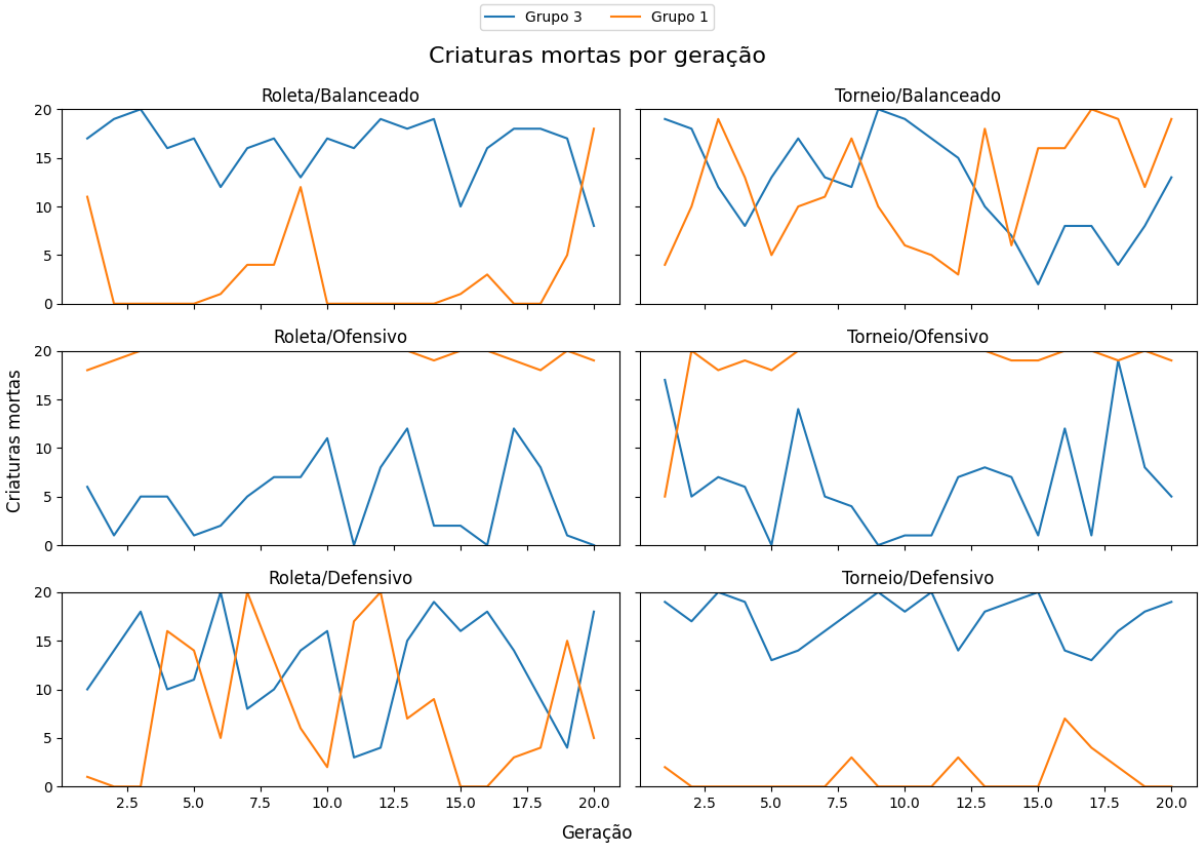


Gráfico 8 – Grupo 1, imutável x Grupo 3, GA: Criaturas mortas.



Ao realizar a observação das armas utilizadas em conjunto com os comportamentos e *traits* usados, ambos os métodos de seleção mostram padrões claros, levando em consideração os estilos de jogo adotados.

Para o método ofensivo, a escolha predominante foi a bomba, que, pela observação do número de criaturas mortas em ambos os métodos de seleção, parece a mais efetiva em eliminar os inimigos. A ausência de outras armas para dano é um sinal de que a bomba foi considerada a melhor arma por um problema de balanceamento. Esse desbalanceamento rendeu os comportamentos escolhidos pelas criaturas ofensivas irrelevantes. Pela predominância da bomba, podemos observar que as criaturas no estilo ofensivo escolheram o *trait MoreExplosiveLessCommonDamage*, enquanto que outros *traits* que buscam diminuir a quantidade de dano não foram escolhidos. Para o estilo defensivo, a escolha para armas era previsível, já que os tiros guiados de cura são a única arma capaz de curar aliados.

Com exceção da bomba e dos tiros guiados de cura, todas as armas restantes foram subutilizadas por ambos os estilos e também métodos de seleção. Apenas o método de roleta balanceado fez uso de granadas e tiros rápidos de dano, mas também de maneira breve. Assim, podemos observar que, dado o desbalanceamento da arma bomba e a capacidade exclusiva de cura de outra arma, as opções das criaturas oscilaram entre essas duas.

Concluímos então que, utilizando um inimigo previsível, as criaturas de todos os estilos e métodos de seleção escolheram opções lógicas que permitiram a adaptação.

Gráfico 9 – Grupo 1, imutável x Grupo 3, GA: Armas utilizadas pelas criaturas geradas proceduralmente por geração.

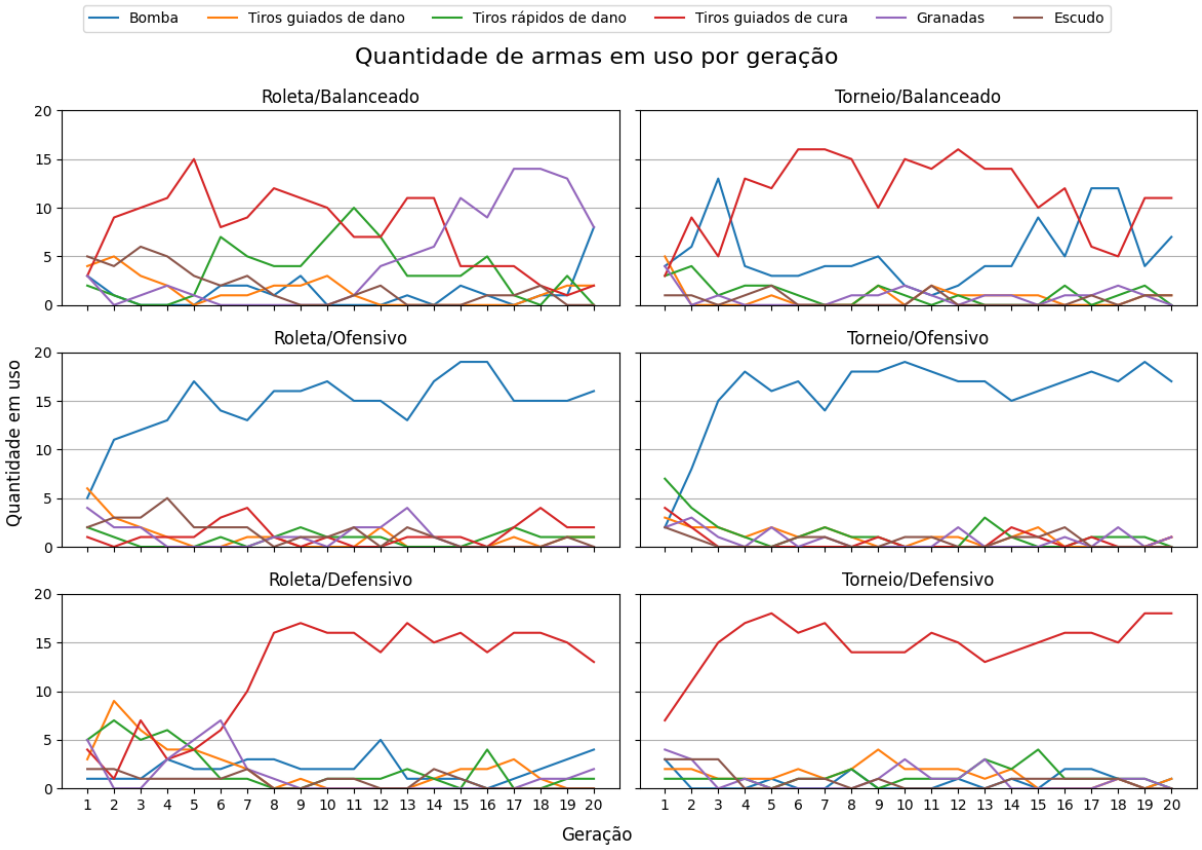


Gráfico 10 – Grupo 1, imutável x Grupo 3, GA: Comportamentos utilizados pelas criaturas geradas proceduralmente por geração.

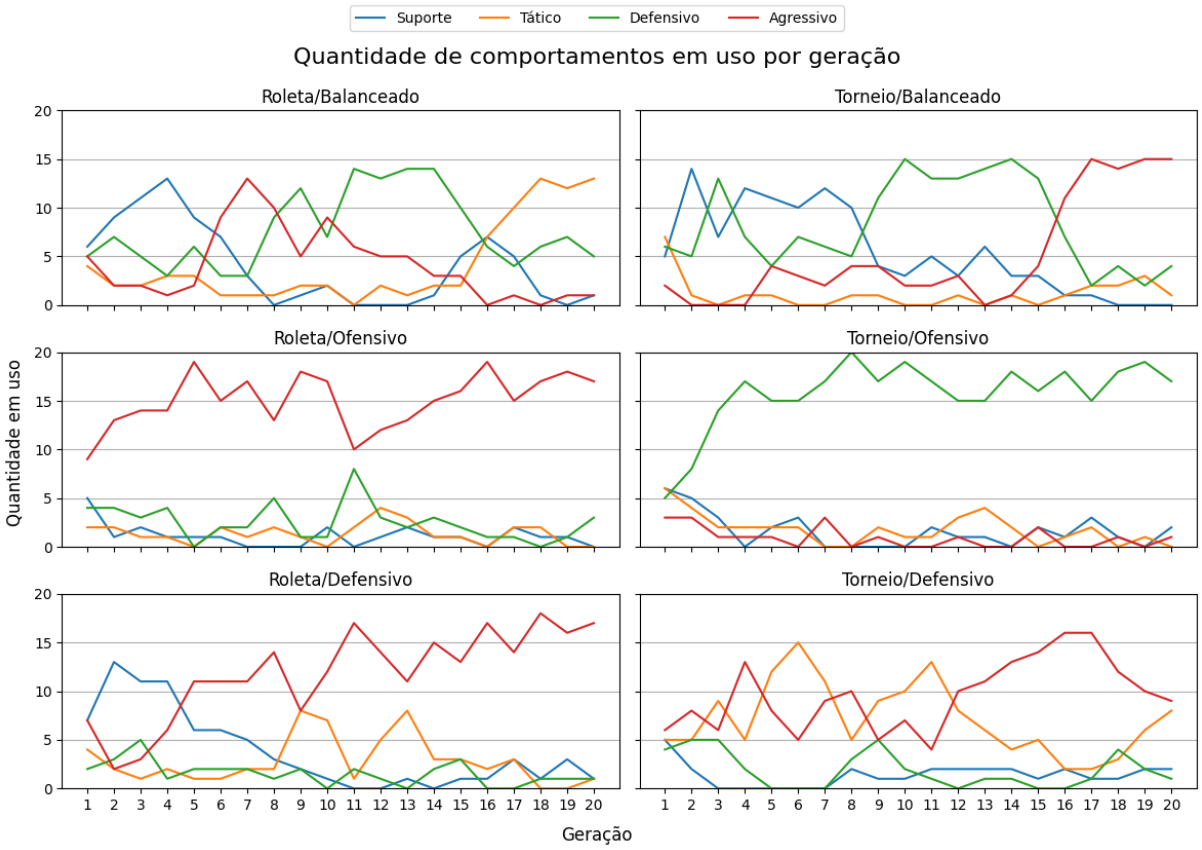


Gráfico 11 – Grupo 1, imutável x Grupo 3, GA: *Traits* utilizados pelas criaturas geradas proceduralmente por geração.

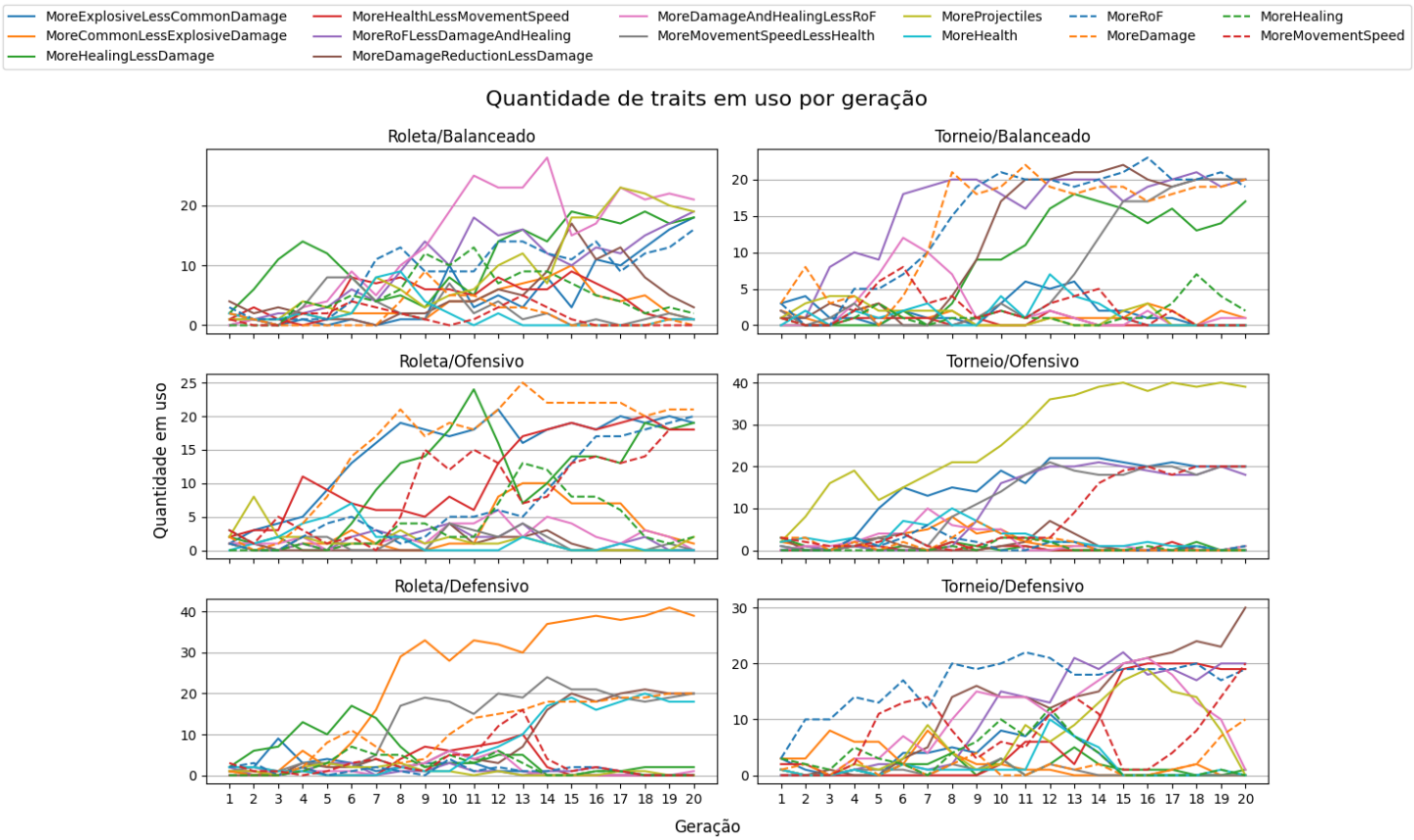
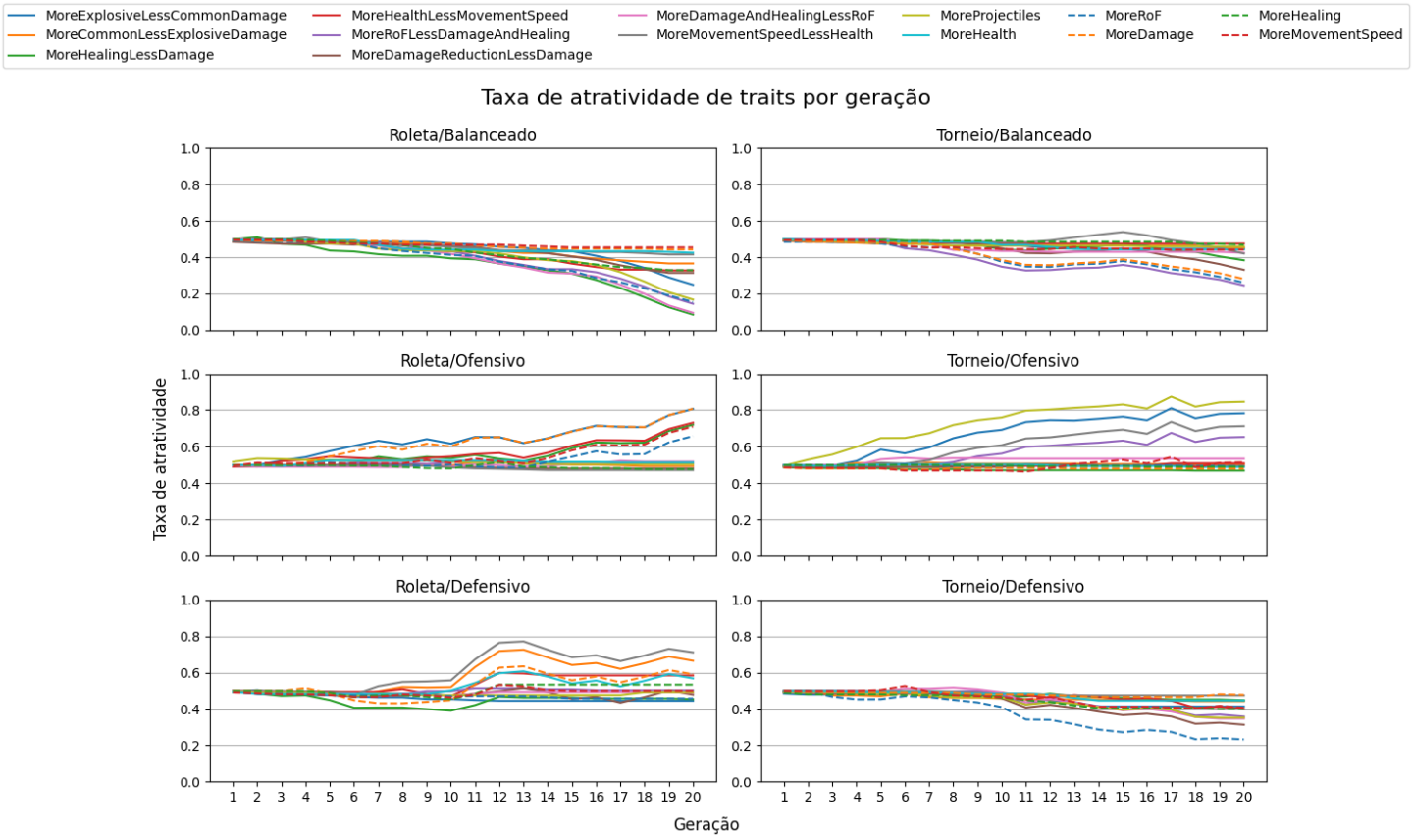


Gráfico 12 – Grupo 1, imutável x Grupo 3, GA: Taxas de atratividade dos *traits* utilizados pelas criaturas geradas proceduralmente por geração.



7.2 GRUPO 2 X GRUPO 3

Assim como no embate do grupo 1, podemos obter uma visão geral sobre o embate através dos gráficos 13, 14, 15.

De maneira similar ao ocorrido no outro embate, os estilos ofensivo e defensivo geraram valores mínimos e máximos de *fitness* mais espaçados, enquanto o balanceado não apresentou diferenças tão significativas.

Nessa seção, a análise dos dados foi quebrada em duas partes, cada uma dedicada para um método de seleção.

Gráfico 13 – Grupo 2, aleatório x Grupo 3, GA: *Fitness* médio das criaturas geradas proceduralmente. (Versus Grupo 2, aleatório)

Gráfico 14 – Grupo 2, aleatório x Grupo 3, GA: *Fitness* mínimo e máximo das criaturas geradas proceduralmente. (Versus Grupo 2, aleatório)

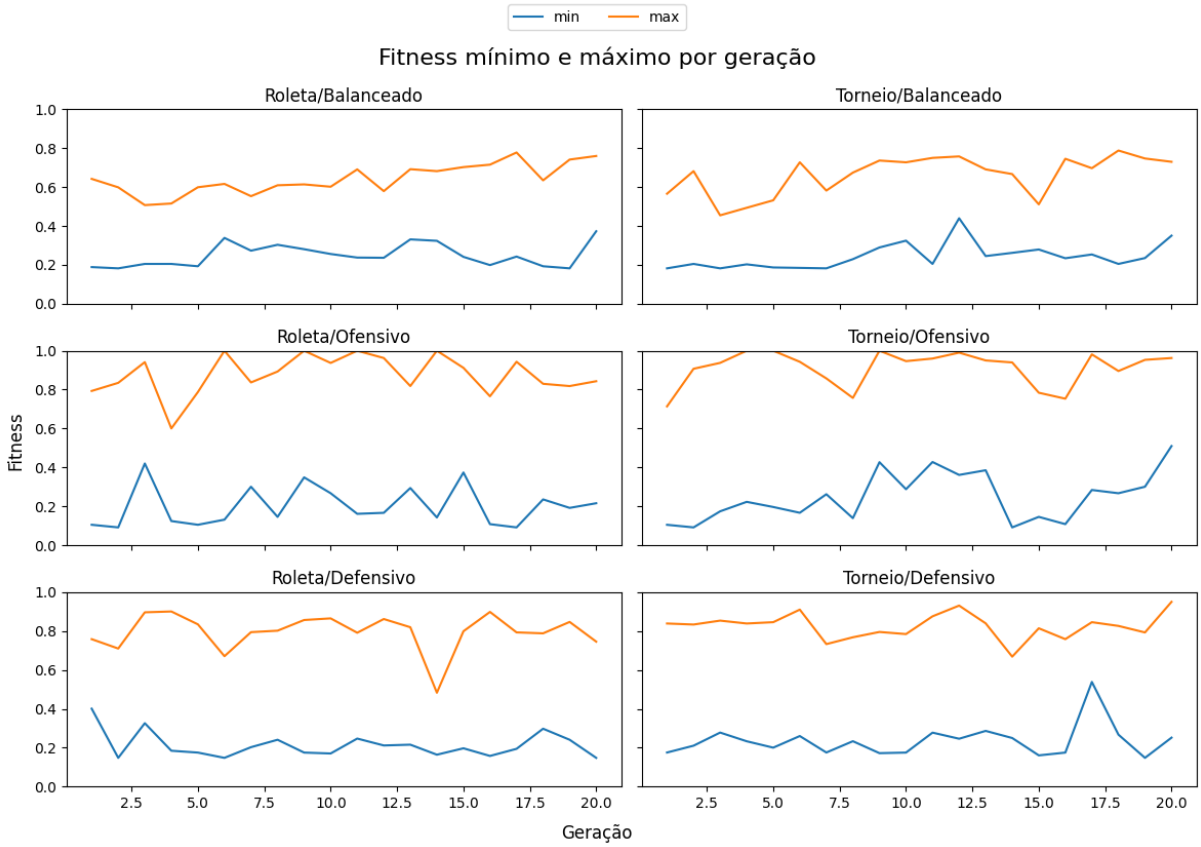
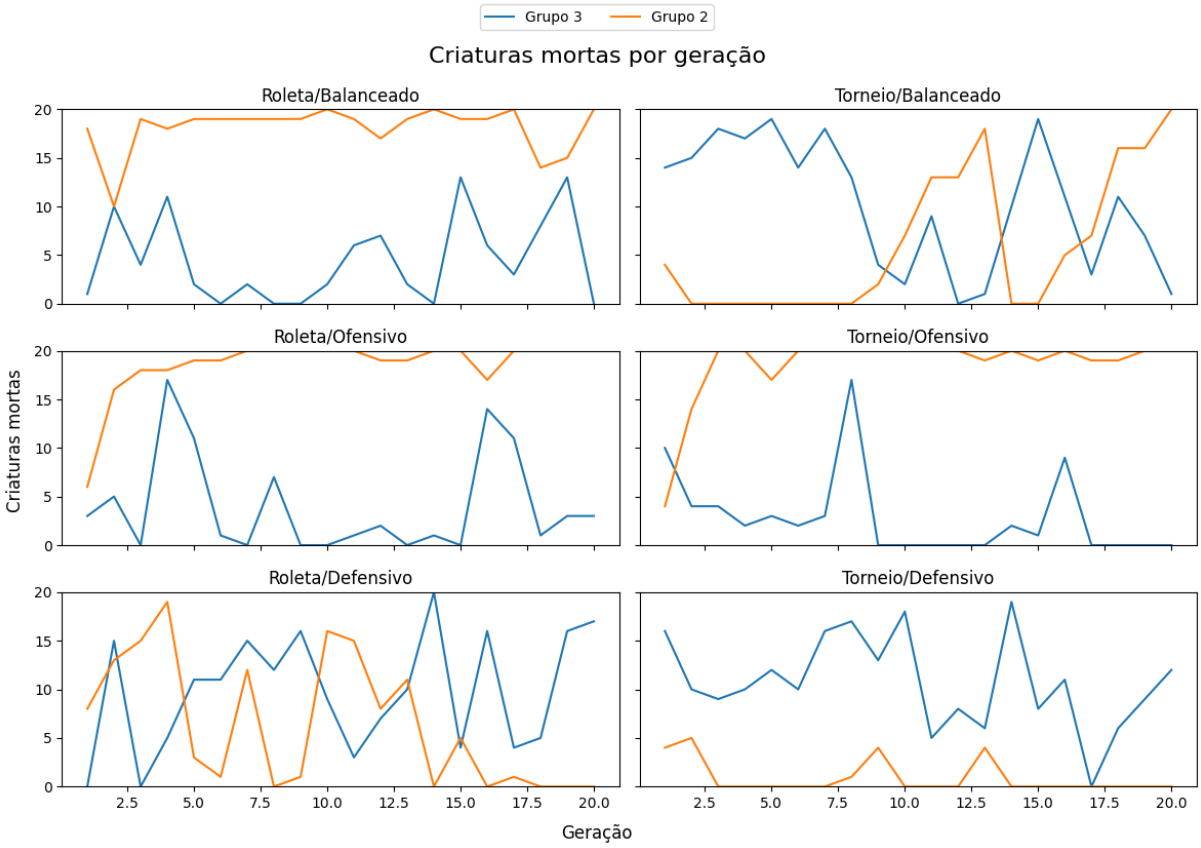


Gráfico 15 – Grupo 2 x Grupo 3: Criaturas mortas.



7.2.0.1 Roleta

No método de roleta, as criaturas do estilo balanceado conseguiram eliminar muitas outras criaturas do que quando comparadas com o embate anterior, do grupo 1.

Ao observarmos a quantidade de armas em uso por geração, encontramos novamente a bomba dominando as outras armas. Assim como no embate anterior, o estilo defensivo escolheu primariamente a única arma capaz de curar.

Esses comportamentos, tendo se repetido entre embates demonstra uma capacidade de adaptação, mesmo quando o seu oponente muda de estilo de jogo constantemente.

No entanto, diferente do outro embate, as escolhas dos *traits* do estilo ofensivo não parece ter sido focada em mais dano, mas com escolhas mais voltadas para velocidade de movimento.

Gráfico 16 – Grupo 2, aleatório x Grupo 3, GA (roleta): Armas em uso por geração.

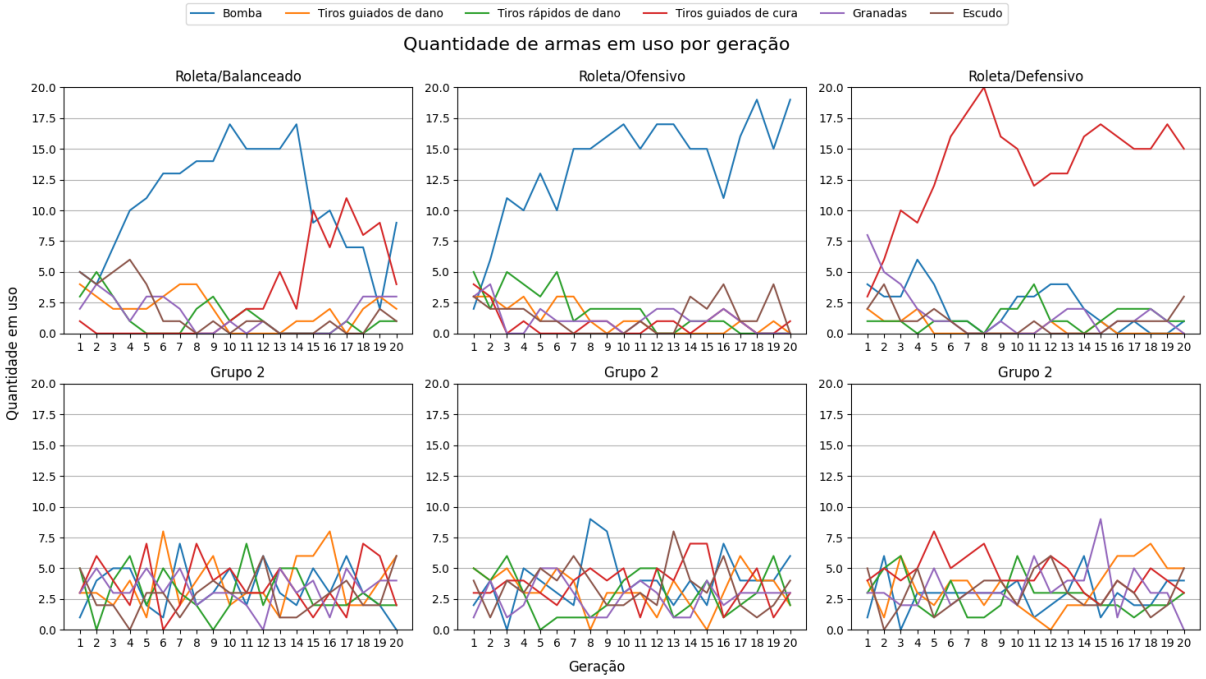


Gráfico 17 – Grupo 2, aleatório x Grupo 3, GA (roleta): Comportamentos em uso por geração.

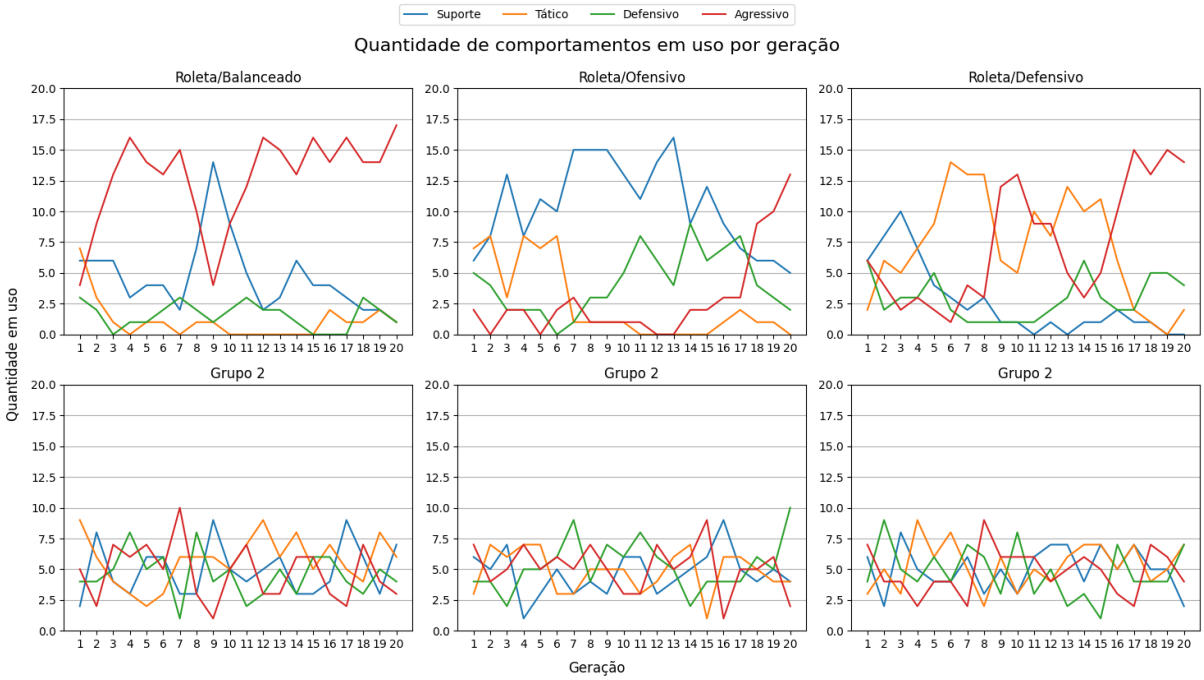
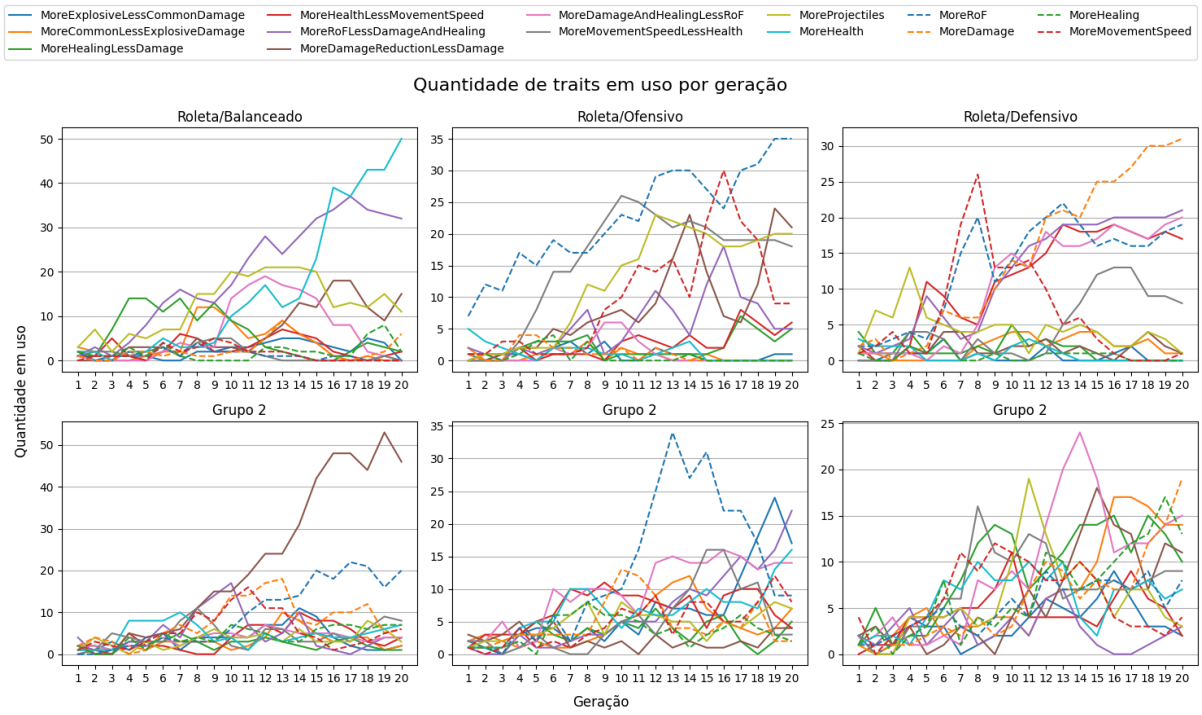


Gráfico 18 – Grupo 2, aleatório x Grupo 3, GA (roleta): *Traits* em uso por geração.



7.2.0.2 Torneio

De maneira similar aos outros embates, novamente temos uma opção clara pela bomba e pelos tiros guiados de cura.

Gráfico 19 – Grupo 2, aleatório x Grupo 3, GA (torneio): Armas em uso por geração.

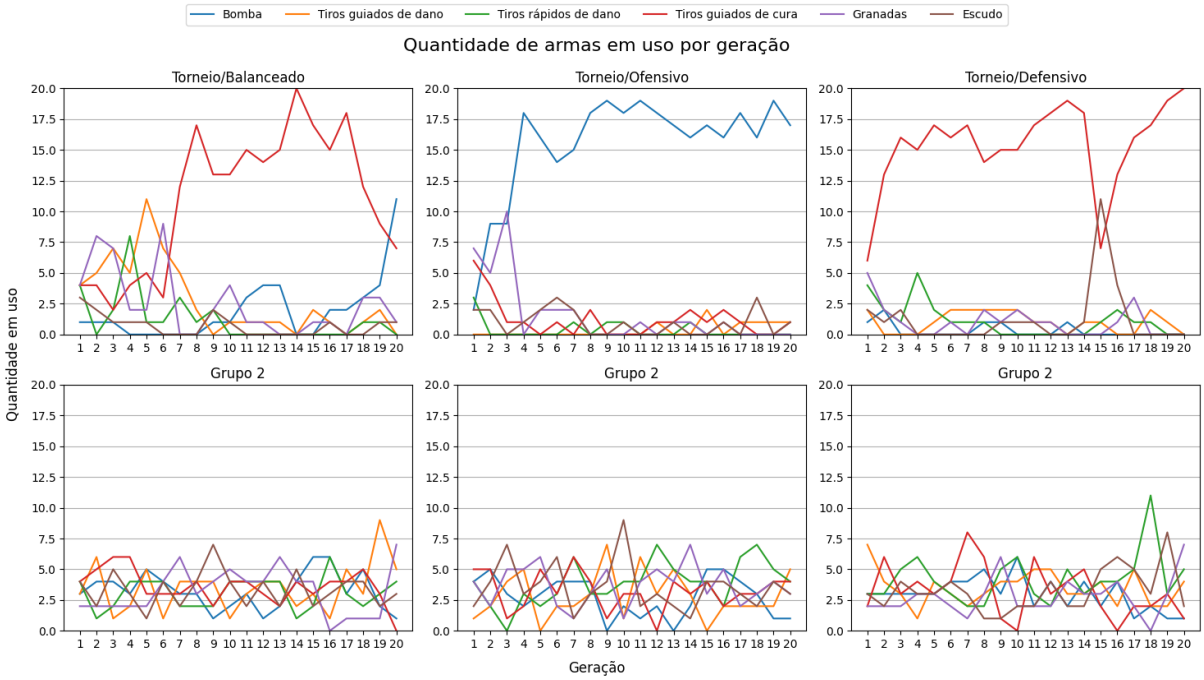


Gráfico 20 – Grupo 2, aleatório x Grupo 3, GA (torneio): Comportamentos em uso por geração.

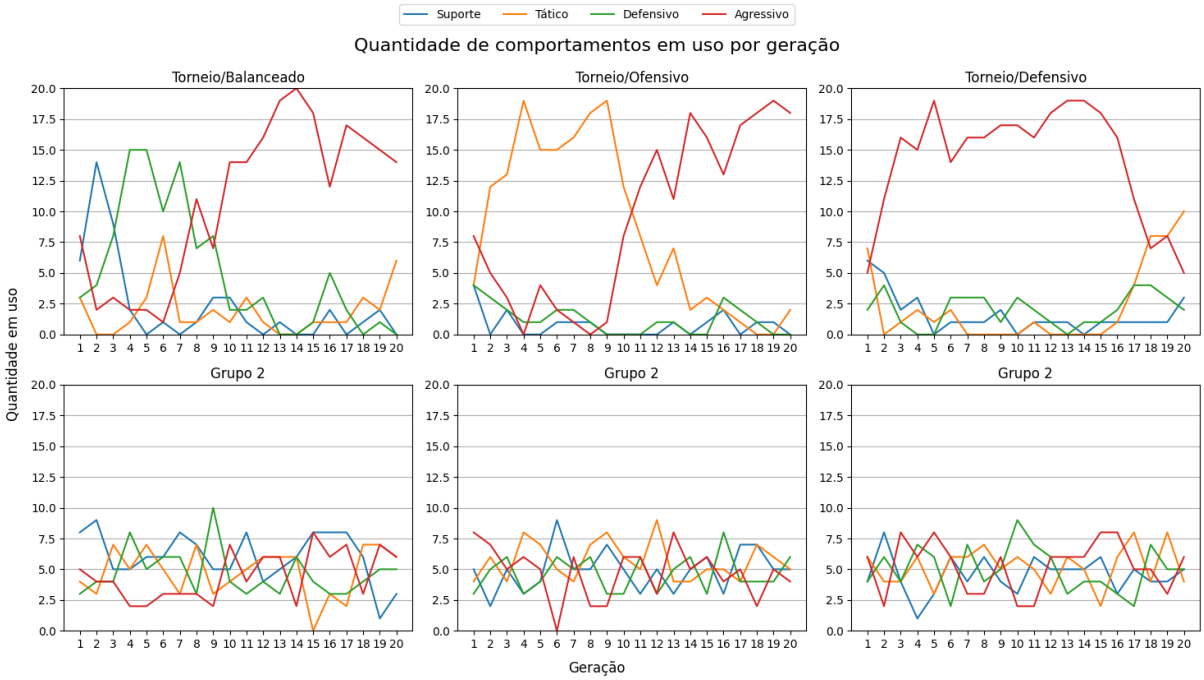
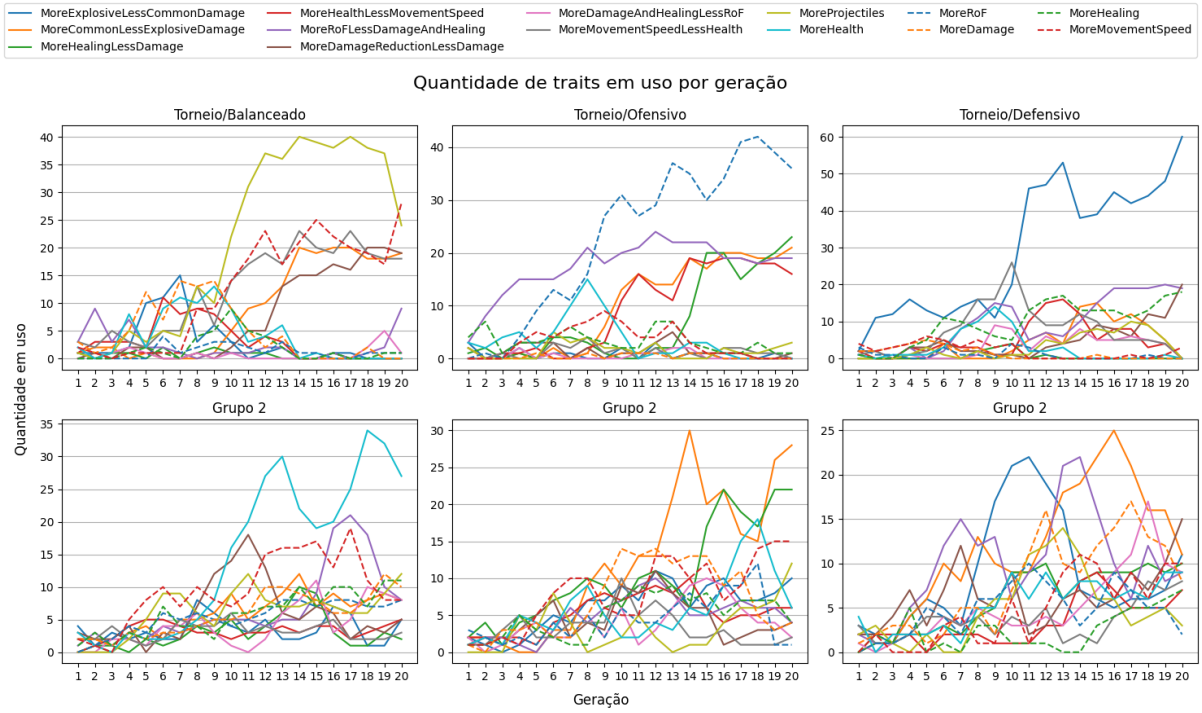


Gráfico 21 – Grupo 2, aleatório x Grupo 3, GA (torneio): *Traits* em uso por geração..



8 CONCLUSÃO E TRABALHOS FUTUROS

Ao longo deste trabalho, foi apresentada a importância da aplicação de PCG para inimigos, juntamente com técnicas para criação de agentes complexos, usando *state machines* e algoritmos genéticos. Durante a pesquisa, o fator de rejogabilidade que serviu como inspiração para o aprofundamento da utilização de PCGs em jogos mostrou-se um fator extremamente complexo, criando possibilidades para trabalhos futuros que buscam investigar a relação de técnicas de PCG com o aumento da rejogabilidade em jogos.

Buscando explorar a geração procedural de inimigos adaptativos, foi desenvolvido um protótipo de jogo no qual grupos de agentes foram colocados em batalha, com o objetivo de avaliar a capacidade de adaptação desses agentes. No protótipo desenvolvido, os agentes eram capazes de obter diversas armas, *traits* - características especiais que alteram seus atributos - e comportamentos, os quais eram ditados pela implementação de uma cadeia de Markov por meio de uma *state machine*.

Ao integrar as mecânicas do jogo com algoritmos genéticos, foi desenvolvido um sistema flexível que permite a adição e remoção de componentes sem comprometer o restante do protótipo. Essa abordagem ampliou as possibilidades de escolha dos genes das criaturas.

Durante os embates, foram coletados dados dos agentes com o objetivo de obter um melhor entendimento das capacidades de adaptação das criaturas geradas pelo algoritmo genético. Os embates ocorreram em duas condições: colocando as criaturas geradas proceduralmente contra inimigos imutáveis, que após uma randomização inicial não sofriam mudanças, e inimigos que eram aleatorizados a cada nova *wave*.

A análise dos dados revelou uma preferência das criaturas geradas por dois tipos de arma, implicando em um desbalanceamento nos valores dessas armas. No entanto, mesmo com essa preferência, as criaturas demonstraram uma notável capacidade de adaptação, optando pelas armas mais poderosas. A escolha dos *traits* foi afetada fortemente pelo combate contra inimigos imutáveis e aleatórios, fazendo com que as escolhas não fossem tão óbvias. Apesar disso, as criaturas geradas proceduralmente apresentaram padrões de comportamento em ambos os casos. As criaturas, avaliadas entre métodos de seleção de torneio e roleta não apresentaram diferenças significativas entre as duas formas de seleção.

No entanto, a capacidade de adaptação dos agentes, embora tenha sido avaliada em combates contra outros agentes, não pode ser exclusivamente definida pelos métodos abordados nesta monografia. Para uma compreensão completa, é necessário colocar os agentes em confronto com jogadores e analisar o impacto que eles possuem no jogo.

Mesmo não sendo capaz de avaliar por completo a capacidade de adaptação dos agentes, o comportamento das criaturas ao priorizar armas poderosas reforça o emprego de GAs para balanceamento em jogos, como demonstrado por Manabe e Miyake (MANABE, 2021).

Para melhor entendermos os motivos que levam a não utilização de técnicas para geração procedural de inimigos na indústria, testes com seres humanos para avaliar o impacto também são extremamente necessários. O projeto desenvolvido para esta monografia demandou um tempo de desenvolvimento alto para a integração de GAs, fator esse fundamental para a análise do custo-benefício da implementação destas técnicas.

Para próximos trabalhos, após os agentes demonstrarem sua capacidade de adaptação em um ambiente isolado de testes, é essencial submetê-los a desafios reais, enfrentando jogadores humanos. Essa etapa permitirá uma avaliação mais abrangente das habilidades adaptativas dos agentes e como eles afetam a experiência do jogo.

Ao confrontar os agentes com jogadores, será possível observar como suas estratégias adaptativas se ajustam aos estilos de jogo e táticas empregadas pelos jogadores humanos. Além disso, dados podem ser coletados a respeito da eficácia dos agentes em diferentes situações de jogo e como eles se comparam aos desafios proporcionados por jogadores reais. Portanto, para uma compreensão completa da capacidade de adaptação dos agentes, é fundamental conduzir testes em um ambiente de jogo realista, envolvendo jogadores humanos como oponentes.

Também é possível pesquisar a aplicação de GAs em diversos jogos *open-source*, podendo obter resultados que busquem entender a baixa utilização na indústria, já que a dificuldade de integração quando o projeto não foi desenvolvido com esse propósito também poderá ser avaliada. Da mesma forma, outras técnicas de *IA* também podem ser aplicadas para a geração procedural, oferecendo comparativos de dificuldades de implementação e resultados obtidos.

REFERÊNCIAS

FONT, J. Evolving third-person shooter enemies to optimize player satisfaction in real-time. In: . [S.l.: s.n.], 2012. v. 7248, p. 204–213. ISBN 978-3-642-29177-7.

HASSANAT, A. et al. Choosing mutation and crossover ratios for genetic algorithmsa review with a new dynamic approach. **Information**, v. 10, n. 12, 2019. Disponível em: <https://www.mdpi.com/2078-2489/10/12/390>.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Ann Arbor: The MIT Press, 1992. 205 p.

HUIZINGA, J. **Homo ludens**. London: Routledge Kegan Paul Ltd, 1949. 220 p.

JEBARI, K. Selection methods for genetic algorithms. **International Journal of Emerging Sciences**, v. 3, p. 333–344, 12 2013.

KAZIMIPOUR, B.; LI, X.; QIN, A. K. A review of population initialization techniques for evolutionary algorithms. In: **2014 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2014. p. 2585–2592.

KRALL, T. M. J. Aspects of replayability and software engineering: Towards a methodology of developing games. **Journal of Software Engineering and Applications**, v. 5, n. 7, p. 459–466, 2012.

LATA, S.; YADAV, S.; SOHAL, A. Comparative study of different selection techniques in genetic algorithm. **International Journal of Engineering Science**, 07 2017.

MANABE, Y. M. K. Game balancing using genetic algorithms to generate player agents. In: **Game AI Pro**. Online edition. [S.l.: s.n.], 2021. cap. 17.

MIRJALILI, S. Genetic algorithm. **Studies in Computational Intelligence**, v. 10, n. 6, p. 43–55, 2019.

OLSSON, V. **A search-based approach for procedurally generating player adapted enemies in real-time**. 2019. 19 f. Monografia (Trabalho de Conclusão de Curso) — Malmö universitet/Teknik och samhälle, Nordenskiöldsgatan 1, 211 19 Malmö, Suécia, 2019.

PEREIRA, L. T.; VIANA, B. M. F.; TOLEDO, C. F. M. Procedural enemy generation through parallel evolutionary algorithm. In: **2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)**. [S.l.: s.n.], 2021. p. 126–135.

SALEN, E. Z. K. **Rules of Play: Game Design Fundamentals**. Cambridge: The MIT Press, 2004. 670 p.

UMBARKAR, D. A.; SHETH, P. Crossover operators in genetic algorithms: A review. **ICTACT Journal on Soft Computing (Volume: 6 , Issue: 1)**, v. 6, 10 2015.

YANNAKAKIS, G. N.; TOGELIUS, J. Experience-driven procedural content generation. **IEEE Transactions on Affective Computing**, v. 2, n. 3, p. 147–161, 2011.

ZHONG, J. et al. Comparison of performance between different selection strategies on simple genetic algorithms. In: . [S.l.: s.n.], 2005. v. 2, p. 1115–1121.

APÊNDICE A – REPOSITÓRIO

O projeto desenvolvido está disponível em um repositório no GitHub, tornando possível a criação de *builds* e acesso ao código para futuros trabalhos. É possível acessar o repositório através do link <https://github.com/LucasSchurer/GeneticAlgorithm>.

NUP: 23081.087782/2023-21 **Prioridade:** Normal

Homologação de ata de defesa de TCC e estágio de graduação

125.322 - Bancas examinadoras de TCC: indicação e atuação

COMPONENTE

Ordem	Descrição	Nome do arquivo
10	Trabalho de conclusão de curso (TCC) (125.32)	Monografia - Lucas Schurer.pdf

Assinaturas

24/07/2023 23:19:08

LUCAS SCHURER MOREIRA (Aluno de Graduação - Aluno Regular)
07.09.12.01.0.0 - Bacharelado em Sistemas de Informação - 121636