

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS  
ESCOLA DE CIÊNCIAS EXATAS E DA COMPUTAÇÃO  
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO



**MIPT: DESENVOLVIMENTO DE UM *WEBSERVICE* QUE GERA MONSTROS  
IDEAIS PARA MESTRES DE *ROLE PLAY GAME* (RPG)**

GABRIEL GIANI REIS

GOIÂNIA  
2021

GABRIEL GIANI REIS

**MIPT: DESENVOLVIMENTO DE UM *WEBSERVICE* QUE GERA MONSTROS  
IDEAIS PARA MESTRES DE *ROLE PLAY GAME* (RPG)**

Trabalho de Conclusão de Curso apresentado à  
Escola de Ciências Exatas e da Computação da  
Pontifícia Universidade Católica de Goiás como parte  
dos requisitos para a obtenção do título de Bacharel  
em Engenharia de Computação.

Orientador: Ms. Fernando Gonçalves Abadia

GOIÂNIA  
2021

GABRIEL GIANI REIS

**MIPT: DESENVOLVIMENTO DE UM *WEBSERVICE* QUE GERA MONSTROS  
IDEAIS PARA MESTRES DE *ROLE PLAY GAME* (RPG)**

Este trabalho de Conclusão de Curso, julgado adequado para obtenção do título de Bacharel em Engenharia de Computação, e aprovado em sua forma final pela Escola de Ciências Exatas e da Computação da Pontifícia Universidade Católica de Goiás, em \_\_/\_\_/\_\_\_\_.

---

Profa. Ma. Ludmilla Reis Pinheiro dos Santos  
Coordenador(a) de Trabalho de Conclusão de Curso

Banca examinadora:

---

Prof. Me. Fernando Gonçalves Abadia

---

Prof. Me. Rafael Leal Martins

---

Prof. Me. Eugenio Júlio M. C. Carvalho

GOIÂNIA  
24 de maio de 2021

## RESUMO

Durante uma campanha de *Role Play Game* (RPG) de mesa será necessário, em algum momento, inserir um combate para os aventureiros enfrentarem. Esse desafio caberá ao mestre, que montará toda a cena, adicionando inimigos a serem enfrentados ao longo da partida. Ademais, para assegurar um momento mais divertido aos jogadores, verifica-se que o mestre deverá alterar algumas características dos monstros presentes no livro base ou criar um monstro “do zero”, pois o balanceamento da criatura é indispensável nesse contexto. Partindo dessa premissa, o presente projeto propõe, a partir da utilização do algoritmo genético e da teoria do *flow*, a criação de um *webservice* capaz de gerar esses inimigos ideais. Para que isso fosse possível, foi desenvolvida uma API (*webservice*) que, ao receber um grupo de aventureiros como entrada, realiza uma chamada para o algoritmo genético que, em seu processo de evolução, devolve uma criatura com estatísticas ideais ao grupo. Vale ressaltar que o algoritmo genético apresentou bons resultados, gerando uma taxa média de 80% de acerto, retornando indivíduos com o nível de desafio necessário para um grupo específico de personagens jogadores. No entanto, para que esse resultado fosse gerado, foram considerados apenas os atributos básicos dos jogadores. Isso equivale a afirmar que esse algoritmo genético não considera habilidades individuais nem magias que o grupo, porventura, possa ter. Além disso, os monstros gerados apresentam as mesmas características de entrada da equipe, ou seja, eles não terão nenhum poder ou habilidade que os tornem mecanicamente únicos.

Palavras-chave: Algoritmo Genético, webservice, RPG, Mestre.

## **ABSTRACT**

During a table RPG campaign, at some point, it will be necessary to insert a combat for adventurers to face. It is up to the master to set up this scene, adding enemies to fight, however, it is often necessary to change some characteristics of the monsters present in the book base or create one from scratch, as the balance of the creature is indispensable for the player experience enjoyment. This project proposes to create a web service capable of generating these ideal enemies using the genetic algorithm and the flow theory. Thus, an API (web service) was required to develop, receiving a group of adventurers as an input, makes a call to the genetic algorithm that in its evolution process returns a creature with ideal statistics to the group. The genetic algorithm showed good results, with an average hit rate of 80%, returning individuals with the level of challenge required for a specific group of the player character, but only basic player attributes were considered for this and, therefore, does not consider skills or spells that the group might have. In the same way, the generated monsters have the same characteristics as the team's input, having no power or ability that makes it mechanically unique.

Key words: Genetic algorithm, webservice, RPG, Dungeon Master.

## LISTA DE ILUSTRAÇÕES

Figura 1 - Modelo de Inimigo .....	20
Figura 2 - Demonstração de um cruzamento em um ponto.....	26
Figura 3 - Fluxograma de um algoritmo genético .....	27
Figura 4 - Possíveis estados do Sujeito na realização de uma atividade .....	29
Figura 5 - Enviando a requisição de consulta de equipes e seu resultado .....	32
Figura 6 - Visual Studio Code com as extensões instaladas e parte do código construído .....	33
Figura 7 - Página do pgAdmin .....	34

## LISTA DE TABELAS

Tabela 1 - Exemplo de Modelos de Animais .....	19
Tabela 2 - Exemplo de Modelos de Construtos.....	19
Tabela 3 - Exemplos de Cromossomos e suas respectivas características .....	24
Tabela 4 - Lista de indivíduos com suas aptidões e sua representação grafica.....	25
Tabela 5 - Configurações extras da máquina virtual utilizada .....	31
Tabela 6 - equipes com suas colunas e descrições, definição e obrigatoriedade da coluna .....	35
Tabela 7 - personagens_jogador com suas colunas e descrições, definição e obrigatoriedade da coluna .....	36
Tabela 8 - monstros com suas colunas e descrições, definição e obrigatoriedade da coluna..	37
Tabela 9 - r_equipes_personagens_jogador com suas colunas, descrições, definição e obrigatoriedade da coluna.....	37
Tabela 10 - Descrição das rotas <i>Index</i> .....	38
Tabela 11 - Descrição das rotas <i>Create</i> .....	40
Tabela 12 - Descrição das rotas <i>Show</i> .....	42
Tabela 13 - Descrição das rotas <i>Update</i> .....	44
Tabela 14 - Descrição das rotas <i>Delete</i> .....	45
Tabela 15 - Relacionamento da dificuldade com as variáveis auxiliares .....	46
Tabela 16 - Período numérico sortido para cada gene do cromossomo .....	47
Tabela 17 - Exemplo de defesas de cromossomos e seus respectivos resultados .....	48
Tabela 18 - Equipe I com jogadores de nível 5 .....	51
Tabela 19 - Equipe II com jogadores de nível 1 .....	51
Tabela 20 - Equipe III com jogadores de nível 15 .....	52
Tabela 21 - Exemplos de cromossomos da Equipe I de nível 5 .....	53
Tabela 22 - Exemplos de cromossomos da Equipe II de nível 1.....	53
Tabela 23 - Exemplos de cromossomos da Equipe III de nível 15 .....	54

## LISTA DE SIGLAS

AG = Algoritmo Genético

API = Application Programming Interface

D20 = Dado de 20 faces

DER = Diagrama de Entidade Relacional

ID = Identificador Único

JSON = JavaScript Object Notation

MVC = Model View Controller

ND = Nível de Desafio

NPC = Non-player character

PJs = Personagens Jogadores

PM = Pontos de Mana

PdM = Personagem do Mestre

PV = Pontos de Vida

REST = Representational State Transfer

RPG = Role Playing Game

SO = Sistema Operacional

T20 = Tormenta 20

TCC = Trabalho de Conclusão de Curso

TED = Technology, Entertainment, Design

URL = Uniform Resource Locator



## SUMÁRIO

<b>1. INTRODUÇÃO</b> .....	<b>11</b>
<b>1.1. Justificativa</b> .....	<b>13</b>
<b>1.2. Objetivos</b> .....	<b>13</b>
<i>1.2.1. Objetivo geral</i> .....	<i>13</i>
<i>1.2.2. Objetivos específicos</i> .....	<i>13</i>
<b>1.3. Metodologia</b> .....	<b>13</b>
<b>1.4. Organização Textual</b> .....	<b>14</b>
<b>2. FUNDAMENTAÇÃO TEÓRICA</b> .....	<b>16</b>
<b>2.1. Cena de combate</b> .....	<b>16</b>
<b>2.2. Algoritmo genético</b> .....	<b>23</b>
<i>2.2.1. Indivíduo e população</i> .....	<i>23</i>
<i>2.2.2. Seleção</i> .....	<i>24</i>
<i>2.2.3. Mutação</i> .....	<i>25</i>
<i>2.2.4. Recombinação</i> .....	<i>25</i>
<i>2.2.5. Função de Avaliação</i> .....	<i>26</i>
<i>2.2.6. Funcionamento</i> .....	<i>27</i>
<b>2.3. Metodologia flow</b> .....	<b>28</b>
<b>3. MATERIAIS E MÉTODOS</b> .....	<b>31</b>
<b>4. DESENVOLVIMENTO</b> .....	<b>35</b>
<b>4.1. API</b> .....	<b>35</b>
<i>4.1.1. Rota Index</i> .....	<i>38</i>
<i>4.1.2. Rota create</i> .....	<i>39</i>
<i>4.1.3. Rota show</i> .....	<i>41</i>
<i>4.1.4. Rota update</i> .....	<i>43</i>
<i>4.1.5. Rota destroy</i> .....	<i>45</i>
<b>4.2. Algoritmo Genético</b> .....	<b>45</b>
<b>4.3. Função de Avaliação</b> .....	<b>47</b>
<i>4.3.1. Avaliação da defesa</i> .....	<i>47</i>
<i>4.3.2. Avaliação da vida</i> .....	<i>49</i>
<i>4.3.3. Avaliação dos ataques</i> .....	<i>49</i>
<i>4.3.4. Avaliação dos danos</i> .....	<i>49</i>
<b>5. RESULTADOS</b> .....	<b>51</b>
<b>6. CONSIDERAÇÕES FINAIS</b> .....	<b>58</b>

<b>REFERÊNCIAS.....</b>	<b>60</b>
<b>GLOSSÁRIO .....</b>	<b>62</b>
<b>APÊNDICE .....</b>	<b>63</b>

## 1. INTRODUÇÃO

Segundo Brauner (2013), o *Role-Playing Game* (RPG) consiste em um jogo narrativo de interpretação de papéis no qual os jogadores assumem personagens fictícios que, juntos, desbravam desafios em um mundo fantástico criado e controlado por um jogador denominado Mestre (ou narrador).

Nesse mundo de faz de conta não há vitória nem derrota. Nos diversos cenários em que ficção e realidade caminham juntas, existe apenas a imaginação potencializada pela criatividade dos jogadores. Além da brincadeira não ter duração predeterminada, as funções e atividades dos integrantes são bastantes distintas, isto é, quem escolhe ser um jogador aventureiro não possui as mesmas atribuições que o mestre, que possui mais responsabilidades ao conduzir a partida (MEARLS, 2014).

Os jogadores aventureiros são aqueles que participam da história como personagens que podem ser criados pelo mestre ou pelos próprios jogadores. Os jogadores aventureiros têm o objetivo de seguir a narrativa do mestre, desvendando mistérios e superando desafios dentro da história (MEARLS, 2014).

Os mestres, que também podem ser chamados de narradores ou de contadores de histórias, têm a atribuição de criar aventuras que serão exploradas pelos jogadores, elaborando desafios tais como: enigmas a serem desvendados, monstros a serem derrotados, tesouros a serem descobertos e quaisquer outros cenários capazes de garantir entretenimento à mesa. Em uma única mesa, poderão ter vários jogadores aventureiros, mas haverá apenas um mestre por jogo (BRAUNER, 2013).

O mestre, objetivando dar continuidade à história, deverá criar toda a estrutura dela, determinando as regras que os jogadores deverão seguir, definindo as consequências de cada ação adotada pelos jogadores aventureiros e realizando o monitoramento de vários personagens secundários que irão interagir com os jogadores, a exemplo de aldeões, chefes de masmorras e de monstros para o combate. No entanto, se apenas essas atribuições fossem levadas à risca, o mestre teria, a cada partida, uma experiência muito pesada e nada divertida. Com o intuito de amenizar essa situação, editoras de livros tais como a Jumbo Editora e a *Wizard Of The Coast* criaram livros com sistemas prontos, que disponibilizaram regras gerais para auxiliar o mestre no controle da seção de jogo. Além disso, esses livros também incluíram um universo narrativo que oferece cidades, povos, culturas e leis próprias desse mundo de fantasia – dessa forma, os mestres têm menos atribuições e mais lazer durante o jogo. Apesar desses livros de regras

possuírem muitas características em comum e compartilhem da mesma ideia, também contemplam regras especiais que tornam cada sistema único.

Dentre os diversos sistemas existentes, para a elaboração do presente trabalho foi escolhido o recente Tormenta 20, conhecido como T20, que é considerado o maior universo de fantasia do Brasil. Apesar de ter sido publicado apenas em 2020, o T20 conta com um rico universo fantástico que começou a ser desenvolvido em 1999 pelos autores Leonel Caldela, Marcelo Cassaro, Rogerio Saladino, Guilherme Dei Svaldi e J.M. Trevisan (SVALDI, 2020).

Ratificando o que foi mencionado alhures, os livros jogos trazem ferramentas de auxílio ao mestre e, dentre elas, merece destaque o Combate, livro de regra básica do T20 que traz o nível de desafio (ND). Esse nível de desafio, segundo autores, consiste numa medida representativa da dificuldade do inimigo e na definição do nível em que o grupo de aventureiros deverá estar para que haja um desafio equilibrado – nesse caso, 4 jogadores aventureiros são considerados um grupo (SVALDI, 2020). No entanto, esse cálculo não garante o equilíbrio para todos os casos, pois essa medida, além de ser geral, não envolve as estatísticas do grupo nem a probabilidade envolvida na cena, uma vez que o combate é decidido a partir da rolagem de dados. Ademais, o livro enfatiza:

Nenhum grupo é igual ao outro. Enquanto alguns jogadores priorizam história e interpretação, às vezes até mesmo fazendo escolhas deliberadamente fracas em termos mecânicos em prol de um conceito, outros são máquinas de otimização, capazes de alcançar valores altíssimos de ataque, defesa e outras características (os “combeiros”). Assim, é impossível garantir que os níveis de desafio do livro funcionem para o seu grupo.

Este projeto tenta suprir o supramencionado problema gerando estatísticas dos inimigos de forma proporcional à equipe de personagens jogadores da partida. Dessa forma, os combates seriam equilibrados porque levariam em consideração os fatores que o ND não analisaria. Vale ressaltar que o método proposto ainda não será capaz de encontrar o equilíbrio perfeito, pois não considerará a probabilidade dos dados que, por sua vez, consiste numa mecânica do combate que não deve ser suprida.

As estatísticas foram calculadas através do método heurístico e evolutivo, o algoritmo genético. Após gerar a população de inimigos, foi utilizado o método *flow* para encontrar o equilíbrio e definir se os monstros estavam proporcionais aos aventureiros.

A seguir, os tópicos apresentarão a justificativa do estudo, os objetivos almejados e a organização textual dos demais capítulos.

## 1.1. Justificativa

A grande dificuldade associada ao tempo que o mestre de RPG empregará na criação de um bom cenário de combate, especialmente naquele em que inimigos e jogadores irão se enfrentar, demandam muito esforço. Tal situação justifica o estudo desse narrador, que inclui a análise de muitos pontos, tais como a força dos jogadores tanto individualmente quanto em equipe, as estatísticas dos monstros, o tipo de criatura, ambientes, dentre vários outros. Apesar de haver modelos prontos de criaturas, na maioria dos casos, os inimigos acabam sendo fortes ou fracos demais, mesmo estando na faixa de nível recomendado.

## 1.2. Objetivos

Este tópico tem como propósito informar ao leitor quais objetivos este trabalho almejou alcançar, dividindo em dois subtópicos: um objetivo geral, centrado no tema proposto, e vários específicos, que definem passos necessários para que o objetivo principal seja obtido.

### 1.2.1. *Objetivo geral*

Desenvolver um *webservice* capaz de gerar inimigos equilibrados em relação a um determinado grupo de aventureiros.

### 1.2.2. *Objetivos específicos*

- Desenvolver um algoritmo genético capaz de gerar as estatísticas dos inimigos;
- Desenvolver um *Flow* que mantenha o nível de desafio na medida ideal;
- Aprimorar os conhecimentos sobre o algoritmo genético e o método *Flow*.

## 1.3. Metodologia

Este trabalho, quanto à sua natureza, é original pois segundo Wazilawick (2014), para ser original, deve gerar novos conhecimentos através de teorias e conhecimentos já existentes. Neste projeto, foi desenvolvido um sistema Web para atingir os objetivos utilizando as técnicas e teorias existentes sobre algoritmos genéticos e os métodos *flows*.

Em relação aos objetivos, este trabalho é considerado explicativo, pois visa clarificar os dados coletados e suas causas e relação. Neste escopo, a pesquisa visa gerar as estatísticas dos inimigos e justificar o motivo deles serem ideais para o cenário criado pelo mestre (WAZILAWICK, 2014).

Quanto aos procedimentos técnicos, o trabalho tem início como pesquisa bibliográfica, pois inicialmente foi feito um levantamento de artigos, revistas e livros na área de conhecimento em estudo para coletar conhecimentos existentes. Como essa técnica não gera conhecimento, apenas suprindo o autor com conhecimentos existentes (WAZILAWICK, 2014), o projeto evolui, tornando-se um trabalho de pesquisa experimental, uma vez que objetiva desenvolver um aplicativo que gera vários testes com base nas estatísticas dos inimigos, analisando-as e selecionando a melhor dentre o grupo.

As áreas pesquisadas são:

- Métodos heurísticos e algoritmos evolutivos, para o algoritmo genético;
- Técnicas e boas práticas na linguagem de programação *Ruby*;
- Estudos de técnicas e conceitos do método *Flow*;
- Técnicas e conceitos para a escrita correta de um Trabalho de Conclusão de Curso (TCC)

#### **1.4. Organização Textual**

Os capítulos seguintes foram divididos em fundamentação teórica, materiais e métodos, desenvolvimento, resultados e considerações finais.

No capítulo a seguir, será apresentada uma contextualização teórica dos conceitos aplicados neste trabalho, estudo que se faz necessário para melhor compreensão acerca do projeto.

Os materiais e métodos informam ao leitor quais programas foram necessários para o desenvolvimento da *Application Programming Interface* (API), citação dos principais autores que contribuíram e como foi configurado o ambiente de desenvolvimento.

Enquanto o capítulo anterior apresentou quais meios foram utilizados para desenvolver o *software*, no desenvolvimento serão exibidos os realizados feitos para a criação em si, com demonstração dos códigos construídos para o algoritmo genético, a API e como foi implementada a lógica da função de avaliação. Na sessão de Resultados, são apresentados os dados gerados pelo projeto.

Por fim, nas considerações finais, serão revistos os objetivos impostos neste projeto, dissertando se foi possível alcançá-los e se houve alguma dificuldade. Afinal, será ventilada a possibilidade de futuros trabalhos derivados do presente projeto.

## 2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, traremos a lume a fundamentação teórica necessária para melhor compreensão do projeto, que está dividido nos seguintes tópicos: cena de combate; algoritmo genético e o método *flow*.

### 2.1. Cena de combate

Em consonância com o capítulo anterior, as regras debatidas neste tópico, embora oferecidas pela maioria dos livros jogos com o intuito de auxiliar os mestres, não são absolutas. O mestre é responsável por gerenciar o jogo e, portanto, tem autonomia de seguir ou de alterar essas regras, ação que é, inclusive, incentivada pelos próprios livros. (MEARLS, 2014; SVALDI, 2020; BRAUNER, 2013)

Durante cenas interpretativas, o narrador gera uma situação em que os jogadores reagem ditando ações de seus personagens. O narrador, então, cita as consequências geradas por essas ações e os jogadores agem novamente, respondendo a esta reação – esse ciclo continua até o fim. Desta maneira, os jogadores têm certa liberdade mecânica, não se limitado aos números de sua ficha, gerando diversas possibilidades geradas pela criatividade e aval do mestre que, por sua vez, diferem de uma cena de combate (BERNACCHIA, 2014).

Uma cena de combate é qualquer situação de confronto entre o grupo de aventureiros contra um ou mais inimigos. Nessa ação, o momento não é passível de ser solucionado a partir de uma simples narrativa, conforme o exemplo a seguir:

“Um grupo de aventureiros se depara com uma carroça tombada e alguns bandidos saqueando esta carroça. Os jogadores então ordenam que os saqueadores parem e se entreguem à guarda local, mas os bandidos se recusam a se renderem e atacam os jogadores.”

A cena é narrativa até o momento em que ocorre o ataque. No exemplo acima, o narrador idealizou um cenário em que os jogadores reagiram ao deparar com uma carroça ocupada por bandidos, ocasião em que deram a ordem de rendição ao passo que o narrador ditou a consequência de ataque contra eles (jogadores). Até esse momento, não foi necessário empregar nenhuma ordem ou regra detalhada, pois tudo foi decidido de maneira interpretativa. Ocorre que, a partir do momento em que o mestre decreta o ataque dos bandidos, surgem alguns elementos que, até então, não eram considerados importantes, tais como: qual indivíduo de qual grupo agirá primeiro? Qual distância estou dos inimigos? Como definir quem acertou e quem errou um ataque? Quantas ações posso fazer durante a minha vez? Essas e outras questões são



resolvidas a partir de uma medição temporal restrita e de regras mecânicas, conforme análise a seguir.

O combate é dividido em várias rodadas. Uma rodada é o tempo que leva para todos os atores envolvidos na cena terminem seus turnos. Um turno é o tempo que cada um tem para poder agir. Um combate pode ser resumido numa sequência de cinco passos:

1. Cada personagem faz um teste para saber quem joga primeiro. Esse teste é realizado a partir da rolagem de um dado de 20 faces (d20) juntamente com a soma da perícia de iniciativa encontrada na ficha de cada jogador;
2. Nem sempre os dois lados do combate estão cientes da batalha, uma vez que o jogador ou o grupo pode ser pego de surpresa no ataque, situação que dará vantagem para o outro time. Em razão disso, o mestre deverá narrar quais personagens estarão (ou não) cientes da cena. Os jogadores surpreendidos ficarão desprevenidos (condição negativa descrita no livro) e, portanto, não terão ação na primeira rodada;
3. Todos os personagens não surpreendidos terão turnos na ordem da iniciativa (do maior para o menor);
4. Quando todos tiverem realizado suas ações, a rodada chegará ao fim, dando início a uma nova rodada, ocasião em que os personagens terão novas ações seguindo a aludida ordem. Entretanto, nessa nova ação, todos poderão agir e essa etapa será repetida até o fim do combate;
5. O combate chegará ao fim quando o objetivo for concluído, ou seja, quando o inimigo for derrotado, se render ou quando qualquer outra condição imposta pelo mestre for realizada.

O jogador poderá, durante o seu turno, realizar as seguintes ações: ação padrão; ação de movimento; ação completa; ação livre e reação. Isso equivale a afirmar que, durante seu turno, o jogador poderá realizar uma ação padrão e uma ação de movimento, duas ações de movimento ou uma ação completa. Nesse ponto, vale salientar que as ações livres e de reações são ilimitadas.

A ação padrão confere ao jogador a possibilidade de executar uma tarefa que requer esforço para sua conclusão, o que equivale a afirmar que haverá, por exemplo, a possibilidade de atacar o inimigo, causando dano, ou de realizar uma manobra de combate que consistirá em um ataque com intuito diferente de causar dano, tal como arrancar a arma do inimigo ou derrubá-lo ao chão.

A ação de movimento serve para mudar algo de lugar. “Algo”, nesse contexto, poderá significar o próprio jogador, enquanto personagem, ou um objeto qualquer. Portanto, se movimentar para algum ponto, desembainhar uma arma ou pegar um item da mochila são ações de movimento.

Uma ação completa de um jogador requer muito esforço e demanda tempo, gastando um turno completo para sua realização. Exemplo disso é uma corrida que confere ao jogador a possibilidade de deslocamento além do normal em razão da perícia de atletismo. Da mesma forma, algumas magias demandam tempo de execução maior que o padrão e, por esse motivo, também utilizam uma ação completa.

Ações livres e de reações demandam pouco ou nenhum tempo, esforço ou atenção, podendo ser realizadas a qualquer momento e em qualquer quantidade ao longo do turno do jogador. A diferença entre elas é que a ação livre consiste em um movimento consciente do jogador, tal como falar, largar um item ou se jogar ao chão, enquanto a reação equivale a uma resposta ou reflexo automático em relação a uma ação de terceiros. Um teste de reflexos para escapar de uma bola de fogo ou perceber um inimigo escondido são exemplos de reações.

Os jogadores deverão se preocupar com as supramencionadas regras. O mestre, entretanto, terá outras funções, a exemplo do controle do combate, da narração dos objetivos concluídos e do controle realizado sobre os personagens do mestre (PdM). Vale ressaltar que, para que as ações do mestre sejam realizadas, será necessário, antes de qualquer outra coisa, que a partida seja criada.

Para construir o combate, o mestre precisará decidir o inimigo que fará parte da cena. O livro base traz alguns adversários prontos na sessão de ameaças, mas por ser um sistema novo, traz poucos modelos em comparação com a edição anterior ou com o RPG mais jogado, qual seja, o *Dungeons and Dragons*. Caso não seja encontrado um inimigo que encaixe na cena planejada, o mestre terá que adaptar uma criatura de outro sistema ou criar uma criatura do zero.

Independentemente da escolha, o mestre terá o trabalho manual de adaptar as estatísticas ou de gera-las para o sistema alvo. No Tormenta 20, no capítulo do mestre, é fornecida uma receita simples de como criar um *Non-Player Character* (NPC). Primeiramente, são adotados os valores fornecidos pelo livro (13, 12, 11, 10, 9 e 8) para, a seguir, ser realizada a distribuição entre os seis atributos das criaturas existentes, sendo elas: força (FOR); destreza (DES); constituição (CON); sabedoria (SAB); inteligência (INT) e carisma (CAR). Depois dessa seleção, é realizada a escolha do tipo a que pertence a criatura, bem como do nível em que ela se encontra. A lista de modelos base pode ser encontrada no capítulo de ameaças do

livro. Esses modelos trazem quantos pontos de vida, mana (atributo para liberação da magia), perícias e habilidades que da criatura, conforme mostrado nas tabelas 1 e 2 a seguir:

Tabela 1 - Exemplo de Modelos de Animais

Animais	Bestas e feras irracionais sem poderes mágicos;
Pontos de Vida	4 + modificador de constituição por nível;
Pontos de Mana	0;
Perícias	Fortitude, Reflexos e outras 2;
Habilidades	Valor de inteligência 1 ou 2; Visão na penumbra.

Fonte: Tormenta20, 2020


Tabela 2 - Exemplo de Modelos de Construtos

Construtos	Objetos animados ou criaturas artificiais;
Pontos de Vida	5 + modificador de Constituição por nível;
Pontos de Mana	1 por nível;
Perícias	2;
Habilidades	Imunidade e doenças, fadiga, sangramento, sono e veneno; não precisa respirar, alimentar-se e dormir; não recupera pontos de vida por descanso e efeitos de cura; visão no escuro.

Fonte: Tormenta20, 2020

Caso seja necessário, após ajustes de estatísticas, o inimigo receberá equipamentos tais como armas, armaduras, poções ou até tesouros. Após finalizar esses passos, o adversário estará criado. Um exemplo de inimigo será mostrado na figura 1.

Figura 1 - Modelo de Inimigo

<p><b>ARANHA GIGANTE</b> <b>ND 2</b></p> <p>Grandes como cavalos, estas aranhas capturam suas vítimas com teia, disparando-a ou tecendo uma armadilha em alguma passagem, para então paralisá-las com a picada venenosa.</p> <p><i>Monstro 7, Grande</i></p> <hr/> <p><b>INICIATIVA</b> +7, <b>PERCEÇÃO</b> +3, visão no escuro  <b>DEFESA</b> 19, <b>FORT</b> +8, <b>REF</b> +11, <b>VON</b> +3  <b>PONTOS DE VIDA</b> 77  <b>DESLOCAMENTO</b> 12m (8q), escalar 12m (8q)</p> <hr/> <p><b>PONTOS DE MANA</b> 7  <b>CORPO A CORPO</b> Mordida +12 (1d8+5 mais veneno).  <b>TEIA (PADRÃO, 2 PM)</b> A aranha gigante dispara teia em uma área de 3m de lado em alcance curto. Criaturas na área ficam enredadas (Reflexos CD 17 evita). Uma criatura enredada pode se soltar com uma ação completa e um teste de Força ou Acrobacia (CD 20) ou cortando a teia (cada espaço de 1,5m de teia tem 5 PV e RD 5). Fogo queima a teia em duas rodadas (e liberta as criaturas), mas causa 1d6 pontos de dano de fogo por rodada a todas as criaturas nela. A aranha gigante também pode usar a teia para cobrir uma área quadrada com 6m de lado. Por sua semitransparência, a teia é difícil de ver (Percepção CD 20) até ser tarde demais. Uma criatura que entre na área fica enredada. A aranha pode andar na própria teia sem se enredar. Ela percebe automaticamente (como se tivesse percepção às cegas) qualquer criatura na teia.</p> <hr/> <p><b>VENENO</b> Condição fraco (Fort CD 18 anula).</p> <hr/> <p><b>FOR 20, DES 19, CON 12, INT 1, SAB 10, CAR 2</b></p> <hr/> <p><b>PERÍCIAS</b> Furtividade +9.  <b>TESOURO</b> 1d4 doses de veneno de aranha gigante (CD 15 para extrair, T\$ 75 cada dose).</p>	
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

Fonte: Esquerda. Svaldi, 2020; Direita. Brauner, 2013.

Depois de criada, deverá ser levado em conta se a criatura está ou não condizente com o ND que o grupo de aventureiros consegue enfrentar. Isso significa que haverá uma análise em relação ao grau de dificuldade do grupo em relação à criatura, que não poderá estar muito acima, frustrando a experiência dos jogadores, nem muito abaixo, a ponto de não gerar um desafio para o grupo de jogadores.

Na construção de um inimigo ideal, o meio termo deverá prevalecer. Em conformidade com a coluna de SVALDI (2020), um combate é considerado equilibrado quando o inimigo consome cerca de 50% dos recursos do adversário. Nesse caso, os personagens jogadores (PJs). Vale reforçar que o combate não tem o objetivo de consumir a metade dos Pontos de Vida (PV) nem dos Pontos de Mana (PM) de cada personagem considerado individualmente na cena e, nesse ponto, SVALDI explica: durante a ação da cena, o combate é contra o grupo todo.

A título de ilustração da situação em tela, em um grupo de aventureiros formado por um curandeiro, um paladino, feiticeiro e um bardo, o curandeiro pode gastar bastante de sua

mana para manter vivo o paladino que, nessa cena, estará à frente, recebendo os ataques da criatura. Nesse caso, ao mesmo tempo em que o feiticeiro está ajudando o grupo, também estará guardando suas magias mais poderosas para o caso de aparecer mais inimigos. De igual maneira, o bardo do time estará arduamente tocando seu alaúde para fornecer os bônus mais variados para toda a equipe. Ao final, caso o curandeiro tenha gastado toda sua energia e o paladino tenha sobrevivido com 20% da sua vida máxima, o feiticeiro ainda terá boas magias guardadas e o combate poderá ser considerado equilibrado, pois terá consumido cerca de 50% dos recursos do grupo, ao passo em que terá oferecido um desafio que, apesar de ter exigido esforço, foi concluído com sucesso.

Uma das maneiras de testar a criatura gerada é a simulação primitiva de combate contra o grupo alvo, a exemplo da chance de ataque de cada jogador e de seu dano médio. O paladino tem um modificador aumentado em 9 na chance de acerto e a média do dano dele é 15. O arcanista tem o modificador de ataque em 5 e 8 de dano médio. O clérigo possui o bônus de ataque em 5 e 12 de dano médio. O bardo, por fim, possui o modificador de ataque em 5 e 8. É ideal que o guerreiro, que possui maior estatística no grupo, consiga acertar a criatura metade das vezes que atacar, pois é o atacante principal. Isso significa que o guerreiro deve acertar a criatura com uma rolagem igual ou superior a 11 no dado d20, ou seja, o inimigo deve ter por volta de 20 de defesa. O curandeiro, por sua vez, precisará tirar 13 ou mais para acertar, ao passo que o bardo e o arcanista, deverão tirar acima de 15 para obter sucesso no ataque. Os PJs mais fracos ainda terão 25% de chance de acertar, considerando, nesse percentual, que um combate dura, em média, de três a quatro turnos, o que conferirá a eles a chance de acertar ao menos um ataque.

Em seguida, deverá ser analisada a vida do monstro, ocasião em que será feita a verificação do dano em relação à sua capacidade de sobrevivência às quatro rodadas de luta. A título de exemplo, considerando o supramencionado grupo de PJs, em que o guerreiro causa, em média, 15 de dano e, levando em conta que ele deve acertar metade das vezes que atacar, o dano médio ponderado será de 7,5 de dano por turno. Para o clérigo, deverá ser considerada, por questão de praticidade, a mesma chance de acerto do guerreiro, isto é, o dano médio ponderado será de 6 e, por fim, tanto o arcanista quanto o bardo causarão, em média, 2 de dano por rodada. Somando esses danos, em uma rodada, a criatura perderá 17,5 de sua vida, 17 arredondado. Como foi considerado um combate com pelo menos quatro rodadas, o inimigo deverá ter em torno de 68 pontos de vida.

No exemplo acima, a simulação foi realizada a partir da vida e da defesa do inimigo. Entretanto, o fator de acerto e dano da criatura, não foram levados em consideração em razão da sua similaridade, ponto que ficou subentendido na análise realizada no presente projeto.

Por questão de praticidade, no exemplo do inimigo gerado, foram testados os atributos básicos dos PJs. No entanto, em uma simulação real, várias outras características do inimigo e dos aventureiros deverão ser levadas em conta. Nessa simulação, o mesmo curandeiro curou o paladino durante o combate, ou seja, o dano do inimigo deve levar em conta que o clérigo pode ou não curar parte do dano que o monstro causou. Além disso, a simulação levou em conta apenas o dano e o acerto físico do feiticeiro e do bardo, sendo que, em uma cena, eles provavelmente dependeriam de magias. Quanto mais real for a simulação, mais variáveis serão envolvidas, aumentando drasticamente a dificuldade.

Considere a seguinte situação ideal: todas as características do monstro e dos PJs foram consideradas e foi possível atingir o equilíbrio entre eles. As magias do conjurador foram consideradas, habilidades curativas e especiais estão incluídas. Infelizmente, ainda não é possível considerar o combate equilibrado, pois além da interação entre PJs e monstro, variáveis externas podem afetar o objetivo, isto é, o equilíbrio.

Como demonstrado por Corte (2020), o ambiente e a circunstância em que o combate é travado consiste em um exemplo de variável externa que pode alterar drasticamente o equilíbrio. Nesse contexto, caso haja um embate contra um dragão vermelho que está preso dentro de uma caverna apertada, situação que o impedirá de utilizar suas asas em relação a um ambiente aberto, haverá uma vantagem para o grupo de PJs, uma vez que em um espaço aberto, esse inimigo poderia fazer investidas e fugir dos ataques corpo-a-corpo, em uma clara situação de vantagem. Outro exemplo de circunstância e ambientação pode ser dado a partir da situação de um antigo mago muito poderoso que, em um ritual, não poderá dar toda atenção aos jogadores, uma vez que o ritual poderá ser desfeito.

Corte (2020) também cita um outro exemplo, qual seja o da experiência dos jogadores em relação ao jogo. Caso o grupo seja veterano e já tenha travado vários combates, esse grupo terá uma noção mais precisa da situação e, conseqüentemente, mais facilidade para montar estratégias, enquanto um grupo de novatos poderá apresentar dificuldades mecânicas ou, por desconhecimento, deixar passar uma vantagem técnica.

Por fim, deve ser mencionada a probabilidade envolvida na mecânica de combate, que nunca permitirá um equilíbrio perfeito. Essa situação permanecerá independentemente do bônus que o PJ ou monstro tenham, uma vez que seu sucesso no ataque nunca será uma garantia, pois é utilizado um d20 na soma final do ataque. Portanto, mesmo um conjurador com o

modificador de ataque em 5 e um guerreiro com o mesmo modificador em 15, contra uma defesa 20 do inimigo, existe a probabilidade de o conjurador acertar o inimigo tirando um valor 15 no dado enquanto o guerreiro pode errar o monstro tirando menos de 5 no dado. Dentre todos os exemplos elencados até o presente tópico, essa será a única variável que o mestre não tentará suprir, pois consiste numa essência e regra básica de um RPG de mesa, seja a cena um combate ou uma narração.

## **2.2. Algoritmo genético**

Os algoritmos genéticos (AGs) são técnicas de busca estocásticas e iterativas, consistindo numa ramificação dos algoritmos evolucionários que constituem modelos computacionais baseados em mecanismos de evolução (COSTA, 2003). Tais mecanismos de evolução, por sua vez, possuem sua fundamentação numa metáfora do processo biológico da seleção natural definida por Darwin, cujo indivíduo mais apto ao ambiente tende a sobreviver gerando descendentes, enquanto indivíduos menos aptos tendem a ser eliminados no processo. (RYE, 2016; LINDEN, 2012).

Essa técnica adota nomenclaturas muito utilizadas na biologia para definir sua estrutura básica. Dados são tratados como indivíduos (ou cromossomos) e os principais métodos são chamados de seleção, recombinação, mutação e função de avaliação (LINDEN, 2012).

### *2.2.1. Indivíduo e população*

Um indivíduo ou cromossomo é a representação de um conjunto de dados que formam uma possível solução que, por sua vez, será trabalhada pelo AG. Cada dado desse grupo é chamado de gene. Um grupo de cromossomos é denominado população enquanto o genótipo consiste nos valores literais que cada gene possui. O fenótipo é a explicação do que o conjunto representa e todos os cromossomos de um AG são voltados para a resolução do mesmo problema. (LUCAS, 2002)

Na tabela 3, são apresentados dois exemplos de cromossomos, sendo que cada um é adotado para um problema diferente. No primeiro indivíduo, o genótipo é uma sequência de números que, por sua vez, consiste na representação literal da solução representada por esse indivíduo. Percebe-se, a partir dessa análise, que não é possível definir, adotando apenas o genótipo como referência, quais os genes deste cromossomo nem o que ele representa para o

problema. Entretanto, com a descrição do fenótipo, será possível definir os seguintes genes: 17, 15, 13, 12, 10, 08. Cada valor dessa definição representará os atributos base de um personagem de RPG. Nesse conjunto, os atributos base do monstro que está sendo testado serão ressaltados pela terceira coluna.

Tabela 3 - Exemplos de Cromossomos e suas respectivas características

<b>Genótipo</b>	<b>Fenótipo</b>	<b>Problema</b>
171513121008	Força (17), Destreza (15), Constituição (13), Inteligência (12), Sabedoria (10), Carisma (8)	Encontrar o monstro com estatísticas ideais
ABCD	Cidade A, Cidade B, Cidade C, Cidade D	Caixeiro Viajante

Fonte: Autor

O segundo cromossomo mostrado na tabela 3 evidencia uma lista de caracteres que, por sua vez, representa um possível caminho entre cidades para resolver o problema clássico do Caixeiro Viajante.

Além das aludidas características, cada indivíduo recebe um grau de avaliação que representa sua aptidão para o problema a ser tratado, ou seja, o quão próximo da solução ideal cada grupo de genes representa. (GUIMARÃES, 2005)

### 2.2.2. Seleção

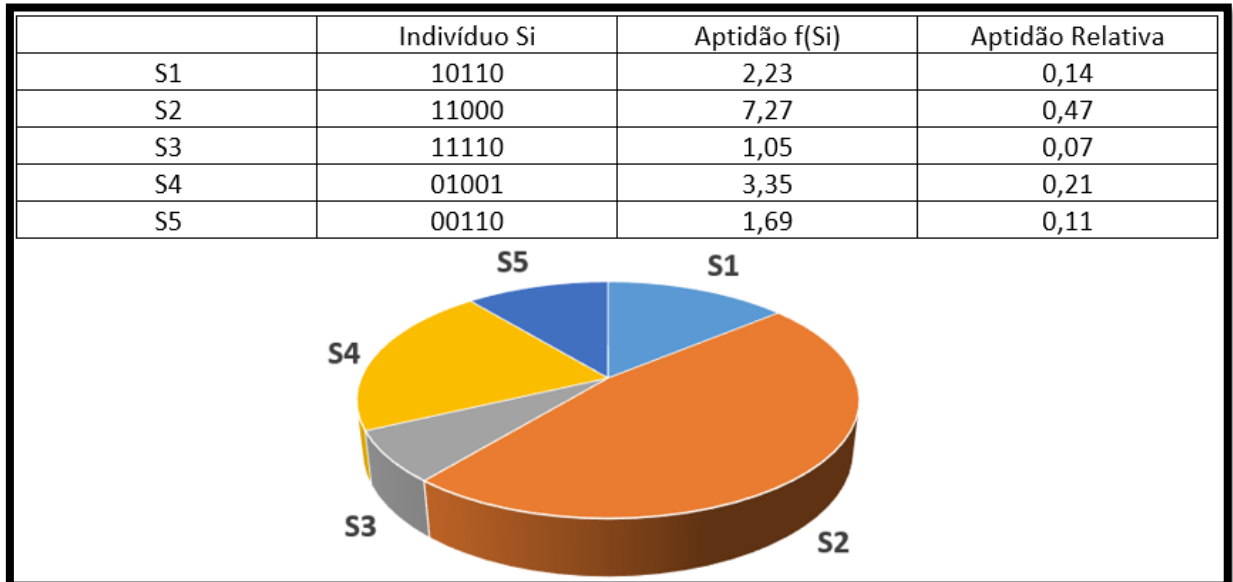
O método de seleção é responsável por escolher quais indivíduos irão passar pelo método de recombinação. Essa escolha é feita de maneira estocástica e diretamente relacionada com o grau de avaliação do indivíduo. Quanto maior for o seu grau, maior será a chance do indivíduo ser escolhido. Vale dizer que até mesmo o indivíduo menos apto participará do sorteio para a seleção, uma vez que nunca haverá um cromossomo com chances nulas (COSTA, 2003).

Costa (2003) ainda afirma que existem diversas maneiras de aplicação do método de seleção. Dentre elas, destacam-se as seguintes: seleção por amostragem universal estocástica, por torneio, por truncagem e por roleta. A título de exemplo, será mostrado o método por roleta, método este escolhido para o AG utilizado no projeto. Neste caso, cada indivíduo da população recebe uma fatia da roleta, sendo esta fatia proporcional ao grau de avaliação dele. Após a inclusão de toda a população na roleta, um número é sorteado e o cromossomo com o número selecionado na fatia é escolhido. O método se repete até que a quantidade de indivíduos



requerido seja selecionado. A figura 2 a seguir, mostra um exemplo de configuração por meio do método da roleta.

Tabela 4 - Lista de indivíduos com suas aptidões e sua representação gráfica



Fonte: <https://sites.icmc.usp.br/andre/research/genetic/>

### 2.2.3. Mutação

A mutação é um dos operadores genéticos responsáveis pela alteração da composição de um indivíduo. Enquanto na etapa de recombinação a transformação do indivíduo ocorre na troca de informações entre a população, na mutação, a transformação é verificada a partir da transformação gene a gene.

Para que a mutação ocorra, cada gene é exposto a uma probabilidade e, caso seja contemplado, seu valor será alterado por outro genótipo possível. Apesar de sua influência aparentar ser pouco impactante devido à baixa probabilidade geralmente atribuída, tem-se um papel fundamental em se evitar o problema do confinamento a ótimos locais. (PATIL, 2014; FILHO, 1998)

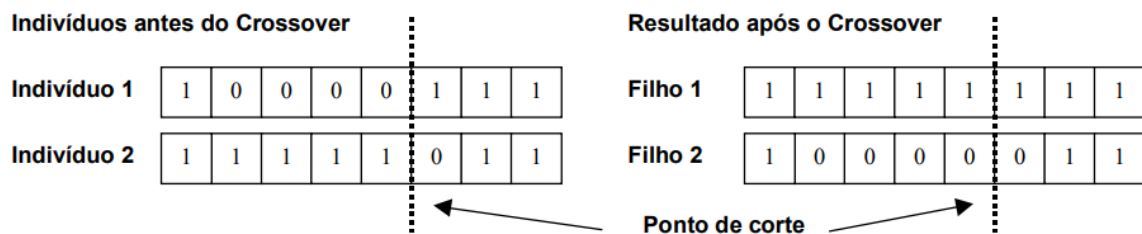
### 2.2.4. Recombinação

A recombinação ou cruzamento é o operador responsável por gerar a nova população de indivíduos. Após a seleção dos indivíduos aptos a este método (pelo operador de roleta), uma dupla de cromossomos é selecionada e eles irão trocar genes entre si, formando outra dupla distinta de cromossomos (comumente chamada de filhos). Esta operação par a par é realizada

até que todos os indivíduos selecionados formem seus descendentes. O método mais simples de recombinação é chamado de cruzamento de um ponto.

A figura 3 demonstra o processo de cruzamento em um ponto. Uma posição qualquer do cromossomo é escolhida de maneira aleatória e aplicada na dupla de indivíduos, dividindo-os em dois e, em seguida, uma permuta é feita entre os dois. O filho I recebe a parte esquerda do segundo cromossomo e a parte direita do primeiro indivíduo. O segundo filho recebe o contrário dessa operação. Com esta operação feita, novos cromossomos são gerados herdando as características de seus antecessores e formando no fim a nova geração de cromossomos. (FILHO, 1998)

Figura 2 - Demonstração de um cruzamento em um ponto



Fonte: Guimarães, 2005

Vale citar que existem outros modelos de cruzamento, tais como cruzamento multiponto, segmentado, uniforme e por combinação parcial, que não serão abordados e nem aplicados neste projeto (LUCAS, 2002).

### 2.2.5. Função de Avaliação

A função de avaliação é responsável por ligar o problema a ser solucionado com o algoritmo genético (ISHIVATARI, 2011). A partir desse método, será possível afirmar se os cromossomos estão convergindo para a solução ou não, atribuindo um grau de avaliação para cada indivíduo.

Considerando que a função de avaliação é um método específico para o problema, cada AG terá uma função distinta, podendo ser algo simples, como uma equação matemática, ou algo mais complexo, com blocos de testes (NETO, 2011). Um exemplo simples, citado por Neto (2011), é um AG que deseja encontrar qual o melhor valor inteiro para  $x$  em uma equação  $3x + 6 = 30$ . Como solução, é possível verificar o quanto  $3x+6-30$  se aproxima ou se distancia

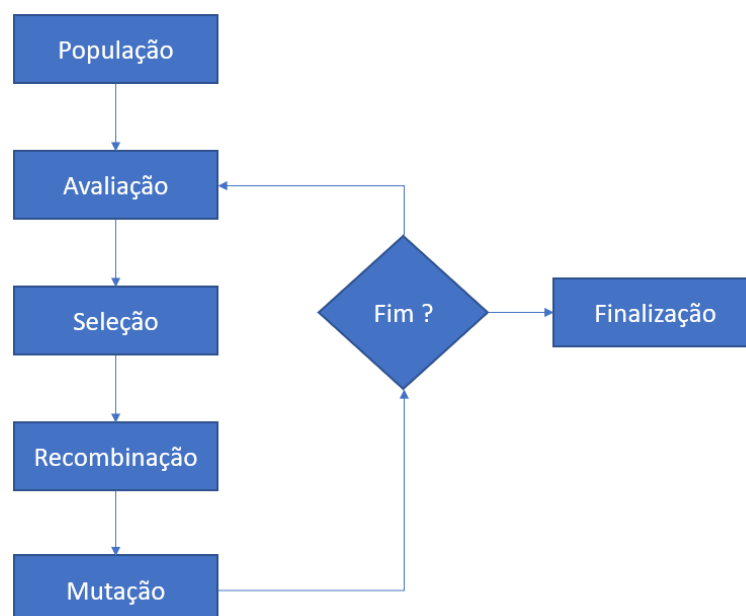
de 0, sendo que  $x$  representa, nessa equação, o cromossomo do algoritmo (FREITAS, 2002; COSTA, 2010). Caso o indivíduo se aproxime, significa que a solução ótima está sendo obtida. Caso contrário, pode ser concluído que está sendo distanciado da solução.

Outro exemplo, um pouco mais complexo para a geração da função, é o problema de encontrar o inimigo ideal apresentado neste projeto, uma vez que a solução não pode ser encontrada com apenas uma linha ou equação matemática, necessitando de uma simulação de combate e um conceito fora do AG, o método *flow*, que gera uma simulação primitiva de combate para verificar se os cromossomos estão ou não equilibrados, ou seja, com a solução ótima.

### 2.2.6. Funcionamento

Conforme demonstra o fluxograma da figura 4, o AG tem início com a criação de vários indivíduos, formando a primeira geração de população e, em seguida, cada cromossomo passa pelo processo de avaliação, recebendo seu grau de aptidão. Após este processo, é usado o método da seleção escolhido no projeto para selecionar os candidatos que irão passar pela recombinação, gerando a próxima população. Os novos indivíduos passam pela probabilidade de terem seus genótipos alterados pela mutação para tentar não cair na solução local (CHAGAS, 2009).

Figura 3 – Fluxograma de um algoritmo genético



Fonte: Autor

Quando os operadores de transformação terminam de agir sobre a nova população, é feita a seguinte pergunta: foi alcançado o critério de parada? Nesse caso, o critério poderá ser a quantidade de iteração alcançada, um indivíduo ter alcançado o valor de avaliação suficiente para o projeto ou outro critério predefinido. Caso a resposta seja positiva, o algoritmo finaliza, apresentando o melhor indivíduo encontrado naquele momento. Caso contrário, ou seja, quando verificada uma resposta negativa, volta para a etapa de avaliação e o fluxograma continua.

### 2.3. Metodologia *flow*

O estudo do *flow* começou em meados de 1990, pelo psicólogo Mihaly Csikszentmihalyi. Neste assunto, Mihaly tenta compreender o que faz um indivíduo alcançar o estado de felicidade, o que torna uma atividade prazerosa ou entediante e porque mesmo sem uma recompensa material ou *status* em troca, uma pessoa se dispõe a fazer tal atividade simplesmente por gostar. Com essas perguntas em mente, Mihaly aprofundou seus estudos sobre psicologia humana e classificou este estado de satisfação pessoal como o estado *flow* (MASSARELLA, 2008).

Csikszentmihalyi descreveu algumas características em comum encontradas nos indivíduos que estavam no estado *flow* durante suas atividades e notou que, mesmo em tarefas distintas, apresentavam os seguintes atributos em comum (CHRISTINA, 2017; KIILI, 2012):

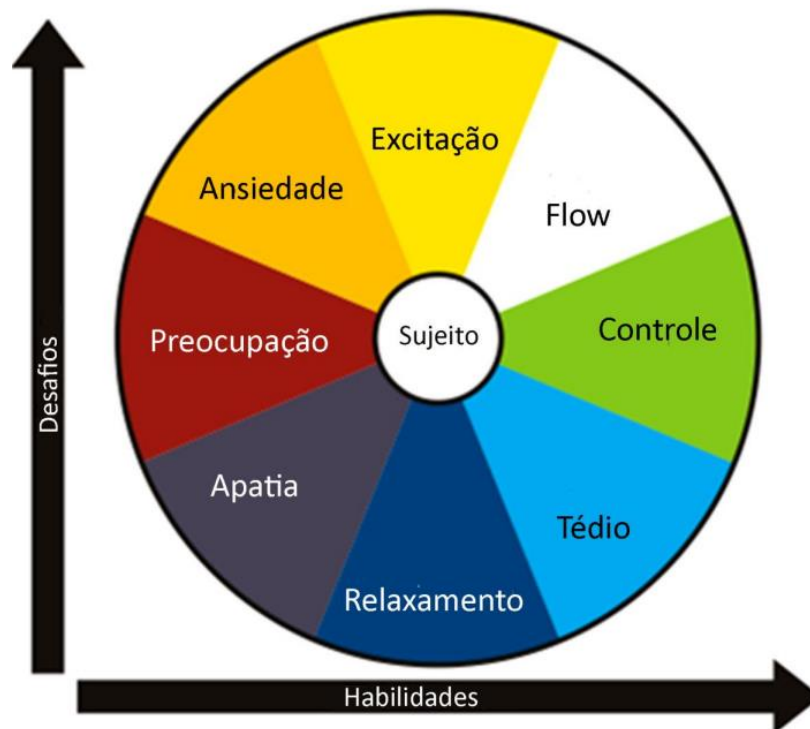
- I. **Foco e concentração:** o indivíduo apresenta envolvimento total com a atividade, dando toda a atenção para a conclusão dela;
- II. **Êxtase:** sensação de estar fora de sua rotina diária;
- III. **Clareza/Feedback:** diretamente ligado com a intensidade na interação do indivíduo com a atividade, gerando um retorno rápido e claro do que deve ser feito para prosseguir com a atividade;
- IV. **Habilidades:** toda atividade tem seu nível de dificuldade e desafio. Esta característica é relacionada ao quão apto o indivíduo está para cumprir o nível de desafio da atividade;
- V. **Crescimento:** aumento gradativo da habilidade no decorrer da atividade;
- VI. **Perda da sensação do tempo:** um foco tão grande que um indivíduo pode passar horas ininterruptas realizando a atividade e quando sai deste estado, a sensação temporal é de ter passado apenas alguns minutos;
- VII. **Motivação intrínseca:** principal característica do estado *flow*, em que a própria realização da atividade é sua recompensa, não se limitando à conclusão.

Lopez (2009) ainda defende que o *flow* é um estado emocional categorizado por Csikszentmihalyi, em que o sujeito está em seu auge, atingindo a melhor performance e satisfação na realização de determinada tarefa, mas não é o único sentimento possível. Como qualquer outra emoção, o *flow* é volátil, podendo sofrer alterações constantes dentre diversos sentimentos durante a realização da tarefa.

Durante uma atividade, seja ela um exercício físico, um jogo eletrônico, ou um estudo, as habilidades do sujeito em concluir os desafios propostos pela atividade deverão estar em constante equilíbrio para verificação do *flow*, pois com os desafios muito altos, a pessoa poderá se sentir frustrada, enquanto com as habilidades demasiadamente altas, poderá sentir indiferença. (CSIKSZENTMIHALYI, 2004)

A figura 5 mostra um diagrama proposto por Csikszentmihalyi (2004), exibindo as diversas emoções presentes e sua relação com os desafios propostos pela atividade, além da habilidade do sujeito para resolver tais desafios.

Figura 4 - Possíveis estados do sujeito na realização de uma atividade



Fonte: Csikszentmihalyi, 2004

Caso o nível de desafio da atividade e a habilidade do sujeito estejam baixas, será gerado o sentimento de apatia (OBADÃ, 2013). Neste momento, o sujeito não tem capacidade para a realização da tarefa, mas fica indiferente ao tentar, pois não existe desafio nem motivação

suficientes para o esforço. Conforme a barra de habilidade cresce, os sentimentos de relaxamento, tédio e controle surgem.

Durante o relaxamento, o sujeito apresenta habilidades médias, suficientes para deixá-lo despreocupado, pois o nível de desafio continua baixo, deixando-o na sua zona de conforto. Ao avançar para o tédio, o indivíduo apresenta proficiência suficiente para negligenciar o desafio, pois não exige nada dele, podendo gerar desânimo e falta de motivação para a conclusão da tarefa, por ser “fácil demais” (CRISTINA, 2017).

De maneira inversa, caso a barra de desafio cresça, o indivíduo sai da apatia, passando pela preocupação, que se torna uma ansiedade. Csikszentmihalyi (2004) discute em sua conferência de *Technology, Entertainment, Design* (TED) que, na etapa de preocupação, o indivíduo tem dificuldade em concluir os desafios, mas ainda vê a possibilidade. Caso o desafio continue a crescer, a ansiedade toma conta e a possibilidade antes vista desaparece. O sujeito é englobado por insegurança e vontade de desistência. Entretanto, quando as barras de desafio e habilidade crescem de forma proporcional, duas emoções próximas do *flow* surgem, quais sejam a excitação e o controle.

Na emoção de excitação, o desafio está levemente acima da capacidade do sujeito, mas por estar próximo da possibilidade de conseguir o restante da habilidade, o desejo de aumentá-la torna a experiência prazerosa para o indivíduo. De maneira contrária, no controle, o sujeito tem sua habilidade um pouco acima do desafio proposto, dando a sensação para o indivíduo de segurança e otimismo para com o desafio proposto, tornando a atividade amigável (LOPEZ, 2009).

Quando o desafio e a habilidade crescem o suficiente, o sujeito encontra dificuldade nos desafios, mas tem a confiança de que conseguirá solucioná-los. Nesse caso, o *flow* surge e o usuário demonstra o melhor desempenho possível (OBADÃ, 2013).

Este projeto necessita de um inimigo que traga o estado de *flow* dos jogadores durante uma cena de combate. O inimigo deverá oferecer desafios suficientes que prendam a atenção dos jogadores e necessitem de estratégias para vencê-lo, mas não poderão ultrapassar as habilidades do grupo, a ponto de tornar a vitória inalcançável. Este é o inimigo ideal.

### 3. MATERIAIS E MÉTODOS

A primeira etapa do projeto foi reunir bibliografias que sustentassem o trabalho realizado, o que tornou este projeto bibliográfico quanto ao seu procedimento. O livro de Svaldi (2020) proporcionou as regras e mecânicas do RPG trabalhado, enquanto as colunas escritas por Della (2020) e Svaldi (2020) na revista *Dragão Brasil* mostraram como os combates do livro base eram calculados e como deixar os inimigos não oficiais equilibrados.

Além dos conceitos relacionados com o mestre de RPG, foi necessário buscar explicações sobre o funcionamento do algoritmo genético e o método *flow*. Para o algoritmo, os autores Linden (2012), Chagas (2009), Ishivatares (2011) e muitos outros foram suficientes para demonstrar o funcionamento e suprir o trabalho com um bom embasamento teórico sobre o assunto, enquanto Csikszentmihalyi (2004), Lopez (2009), Massarella (2008) e outros ofereceram uma base sólida sobre a teoria do *flow*. Com os conhecimentos necessários adquiridos, o projeto evoluiu e, conseqüentemente, os estudos aprendidos foram adotados de maneira prática para alcançar os objetivos definidos o que tornou essa parte do projeto em uma pesquisa experimental.

Na etapa seguinte, foi construído o ambiente de trabalho em que a produção do projeto ocorreu. Para isso, foi utilizada a *VirtualBox* da Oracle. O sistema operacional (SO) instalado na máquina virtual foi o Ubuntu x64, versão 18.04.5 LTS, um SO de código aberto baseado no Debian e tendo o GNOME como ambiente de *desktop*. A tabela 5 mostra os detalhes de configuração deste sistema.

Tabela 5 - Configurações extras da máquina virtual utilizada

RAM	6GB
Chipset	PIIX3
Processador	5CPUs
Disco	43GB

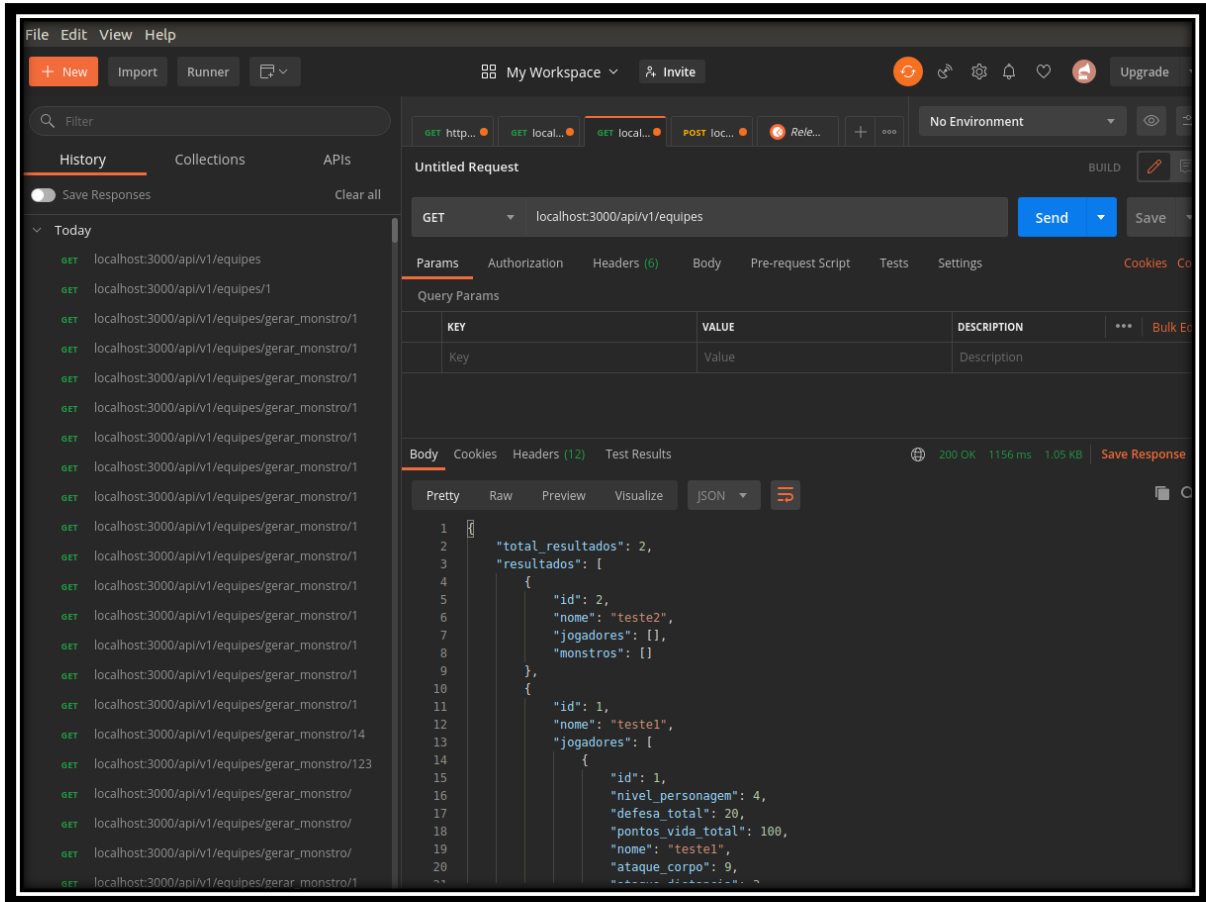
Fonte: Autor

Após a construção do ambiente, os *Softwares* necessários para a produção foram instalados, sendo eles: *Postman*, *Virtual Studio Code* (VSCode), *Postgres*, *pgAdmin*, *Ruby on Rails* (ou simplesmente, *Rails*) e *Ruby*.

O *Postman* é uma ferramenta de auxílio em API que foi utilizada para mandar e receber requisições da API, pois ela não contém interface de comunicação. Vale especificar que a

versão utilizada foi a 7.36.5, distribuição própria. A figura 5 mostra um exemplo de requisição sendo feita para a API e seu resultado.

Figura 5 - Enviando a requisição de consulta de equipes e seu resultado

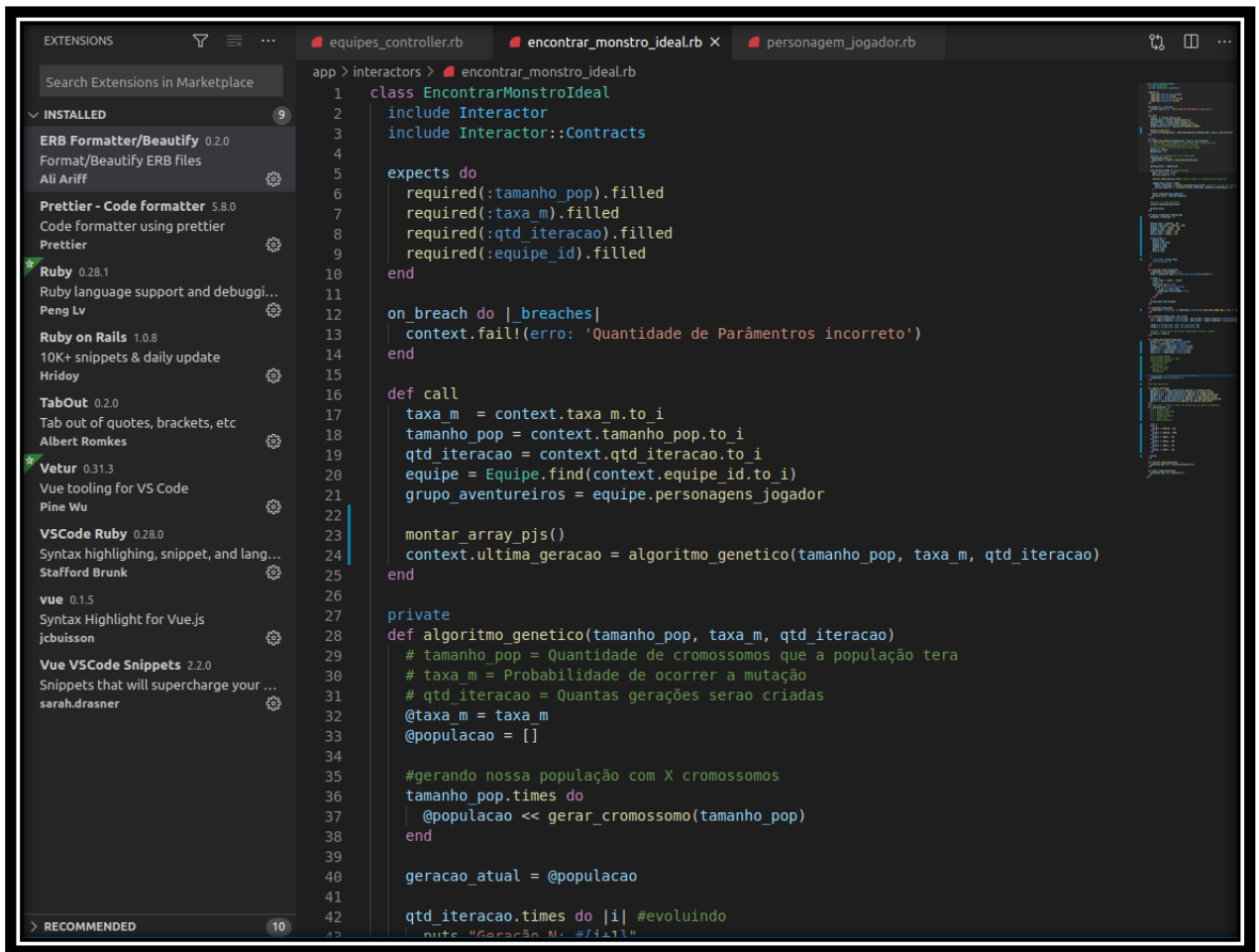


Fonte: autor

O VSCode (versão: 1.48.1, distribuição: Microsoft) é um editor de texto leve. Porém, tal editor é muito poderoso com suporte a várias linguagens de programação, além de oferecer uma vasta biblioteca de extensões feita pela comunidade, desde formatadores, debugadores ou suporte para linguagens extras não suportada em sua versão básica. Esse editor foi a ferramenta principal de codificação. A seguir, na figura 6, são mostradas as extensões instaladas e uma parte da codificação feita.



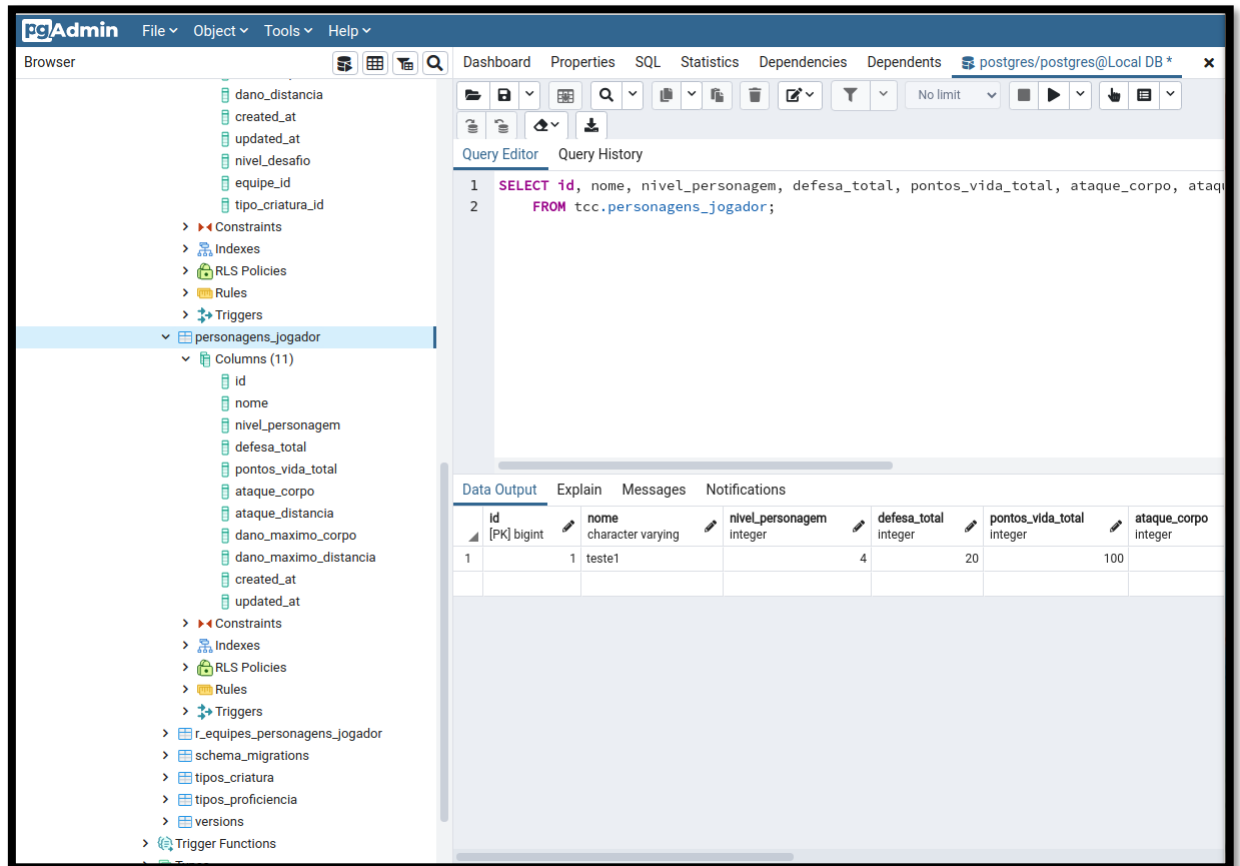
Figura 6 - Visual Studio Code com as extensões instaladas e parte do código construído



Fonte: Autor

Os dados da aplicação foram guardados num banco de dados objeto-relacional, o PostgreSQL (versão: 12.4 distribuição: própria). Para acessar de maneira mais fácil este banco, foi utilizado o pgAdmin (versão: 4.25, distribuição: PostgreSQL), que consiste numa ferramenta de código aberto. A figura 7 mostra uma parte do banco de dados sendo visualizada por meio do pgAdmin.

Figura 7 - Página do pgAdmin



Fonte: Autor

A linguagem escolhida para a construção da API foi o Ruby (versão:2.6.4, distribuição: própria), com o suporte do framework *Rails* (versão: 6.0.3, distribuição: própria).

## 4. DESENVOLVIMENTO

Esta etapa do projeto foi estruturada em três partes principais: a criação da API com suas rotas e *views*, o algoritmo genético e a função de avaliação que, apesar de ser um componente do algoritmo genético, tem importância suficiente para ser um tópico separado.

### 4.1. API

Durante a construção da API, o primeiro passo foi montar o diagrama de entidade relacional (DER) mostrado no apêndice I. Este passo foi fundamental para a elaboração do banco de dados que é consumido pela API, dando uma visão geral do que foi construído. Após a modelagem, foi construído um total de quatro tabelas, sendo três objetos e um relacionamento, quais sejam: *equipes*, *monstros*, *personagens\_jogador* e *r\_equipes\_personagens\_jogador*.

Todas as tabelas apresentam a coluna identificador único (ID), que serve para distinção do dado, contemplando um valor numérico único de referência. Além disso, elas também apresentam, com exceção da tabela *r\_equipes\_personagens\_jogador*, os campos *created\_at* e *updated\_at* que significam, respectivamente, em que data foi criado o dado e em qual data este mesmo campo recebeu atualização.

As tabelas 6 a 9 mostram as colunas restantes, suas respectivas descrições existentes e se o campo é obrigatório ou não.

Tabela 6 - *equipes* com suas colunas e descrições, definição e obrigatoriedade da coluna

<b>equipes</b>		
Armazena todas as equipes geradas no sistema		
<b>Coluna</b>	<b>Descrição</b>	<b>Obrigatoriedade</b>
nome	Campo que representa a nomenclatura atribuída à equipe.	Verdadeiro

Fonte: autor

Tabela 7 - personagens\_jogador com suas colunas e descrições, definição e obrigatoriedade da coluna

<b>personagens_jogador</b>		
Armazena todos os personagens jogadores gerados no sistema		
<b>Coluna</b>	<b>Descrição</b>	<b>Obrigatoriedade</b>
nome	Nome atribuído ao personagem	Verdadeiro
nivel_personagem	Seu nível atual	Verdadeiro
defesa_total	Valor de defesa total	Verdadeiro
pontos_vida_total	Pontos de vida máxima	Verdadeiro
ataque_corpo	Melhor ataque corpo a corpo do jogador	Falso
ataque_distancia	Melhor ataque a distância do Jogador	Falso
dano_maximo_corpo	Dano máximo possível com a arma de ataque corpo a corpo do jogador	Falso
dano_maximo_distancia	Dano máximo possível com a arma à distância do jogador.	Falso

Fonte: autor

Tabela 8 - monstros com suas colunas e descrições, definição e obrigatoriedade da coluna.

<b>Monstros</b>		
Armazena todos os monstros gerados no sistema		
<b>Coluna</b>	<b>Descrição</b>	<b>Obrigatoriedade</b>
nome	Nome atribuído ao monstro	Verdadeiro
nivel_monstro	Seu nível atual	Verdadeiro
defesa_total	Valor de defesa total	Verdadeiro
pontos_vida_total	Pontos de vida máxima	Verdadeiro
ataque_corpo	Melhor ataque corpo a corpo do monstro	Falso
ataque_distancia	Melhor ataque a distância do monstro	Falso
dano_corpo	Melhor dano corpo a corpo do monstro	Falso
dano_distancia	Melhor dano à distância do monstro	Falso
nivel_desafio	Representa o nível de desafio proposto pelo monstro	Verdadeiro
equipe_id	Identificador informando à qual equipe este monstro está relacionado	Verdadeiro

Fonte: autor

Tabela 9 - r\_equipas\_personagens\_jogador com suas colunas, descrições, definição e obrigatoriedade da coluna.

<b>r_equipas_personagens_jogador</b>		
Tabela de relacionamento, serve para conectar vários jogadores com várias equipas (relacionamento n para m)		
<b>Coluna</b>	<b>Descrição</b>	<b>Obrigatoriedade</b>
personagem_jogador_id	Identificador do jogador que está se relacionando	Verdadeiro
equipe_id	Identificador da equipas que está se relacionando	Verdadeiro

Fonte: autor

Em seguida, após a construção do banco, foram desenvolvidas as rotas de acesso da API. Como o modelo utilizado foi o *Model View Controller* (MVC), o *webservice* tem três *controllers* e três *views*, sendo eles: equipes, monstros e personagens\_jogador.

Todos os *controllers*, com exceção de monstros, apresentam as rotas *index*, *show*, *create*, *update* e *destroy*. Monstros não têm as rotas *create* e *uptade*, pois não é possível alterar um monstro após criado e sua criação é apenas a partir do algoritmo genético, não permitindo que a criação seja feita de maneira direta. A seguir, será detalhada cada rota, explicando sua funcionalidade e estruturação.

#### 4.1.1. Rota Index

Esta rota é responsável por buscar todos os dados do *controller* em questão, por exemplo, se é o *index* de equipes, ao chamar a rota, todas as equipes que tiverem cadastrados no sistema serão trazidas. Esta lista é ordenada pelo nome de maneira decrescente.

A tabela 10 a seguir lista as rotas *index* de cada *controller*, juntamente com a *Uniform Resorce Locator* (URL) interna e externa, o protocolo *Representational State Transfer* (REST) utilizado e sua ação.

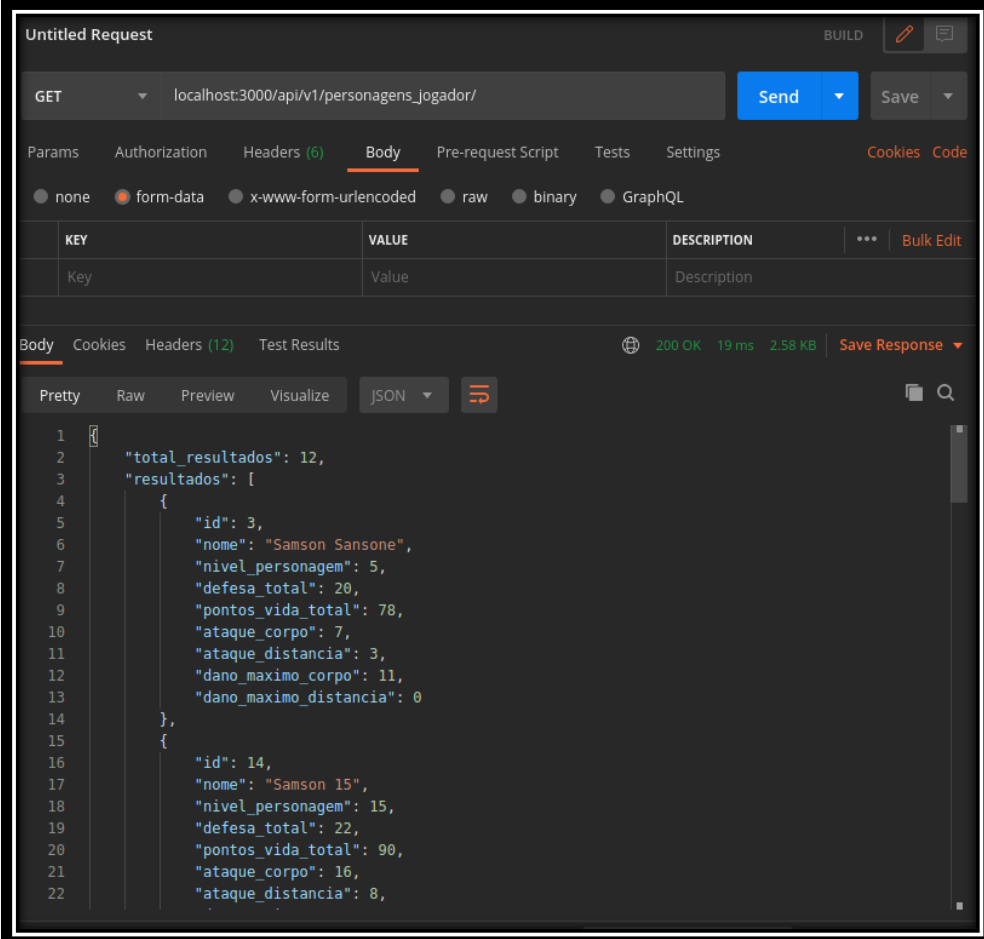
Tabela 10 - Descrição das rotas *Index*

URL interna	URL externa	Protocolo	Ação
api_v1_equipes_path	api/v1/equipes(.:format)	GET	api/v1/equipes#index {:format=>:json}
api_v1_personagens_jogador_path	api/v1/personagens_jogador(.:format)	GET	api/v1/personagens_jogador#index {:format=>:json}
api_v1_monstros_path	api/v1/monstros(.:format)	GET	api/v1/monstros#index {:format=>:json}

Fonte: autor

Quando a rota termina de montar a lista de objetos, ela passa a chamada para a *view* de mesmo nome, *index.json.jbuilder*, que monta em formato *JavaScript Object Notation* (JSON) a saída dos dados. O quadro 1 a seguir mostra um exemplo de chamada da rota feita pelo Postman com sua respectiva saída.

Quadro 1 - requisição feita aos `personagens_jogador` e seu retorno



The screenshot shows a Postman interface for an 'Untitled Request'. The request method is GET and the URL is localhost:3000/api/v1/personagens\_jogador/. The response status is 200 OK, with a response time of 19 ms and a size of 2.58 KB. The response body is displayed in JSON format, showing a list of two character objects.

```

1  {
2    "total_resultados": 12,
3    "resultados": [
4      {
5        "id": 3,
6        "nome": "Samson Sansone",
7        "nivel_personagem": 5,
8        "defesa_total": 20,
9        "pontos_vida_total": 78,
10       "ataque_corpo": 7,
11       "ataque_distancia": 3,
12       "dano_maximo_corpo": 11,
13       "dano_maximo_distancia": 0
14     },
15     {
16       "id": 14,
17       "nome": "Samson 15",
18       "nivel_personagem": 15,
19       "defesa_total": 22,
20       "pontos_vida_total": 90,
21       "ataque_corpo": 16,
22       "ataque_distancia": 8,

```

Fonte: autor

#### 4.1.2. Rota create

Esta rota é responsável por criar o objeto enviado ao sistema, por exemplo, se o *controller* é `personagens_jogador`, esta rota receberá os campos preenchidos pelo usuário e realizará sua inserção no banco de dados. Caso tenha salvado, uma resposta de sucesso será enviada pela API e o usuário será redirecionado a rota *Show*, mostrando o objeto recém-criado. Caso tenha tido algum problema, uma mensagem de erro será mostrada ao usuário com o motivo da falha.

A tabela 11, a seguir, lista as rotas *create* de cada *controller*, juntamente com a *URL* interna e externa, o protocolo REST utilizado e sua ação.

Tabela 11 - Descrição das rotas *Create*

<b>URL interna</b>	<b>URL externa</b>	<b>Protocolo</b>	<b>Ação</b>
api_v1_equipes_path	api/v1/equipes(.:format)	POST	api/v1/equipes#create {:format=>:json}
api_v1_personagens_jogador_path	api/v1/personagens_jogador(.:format)	POST	api/v1/personagens_jogador#create {:format=>:json}

Fonte: autor

Cada *controller* terá os valores de formulário distintos a serem enviados, sendo eles as colunas de sua tabela correspondente. Exemplo: se uma requisição de *personagem\_jogador* estiver sendo feita, os valores aceitos serão *nome*, *nível\_personagem*, *defesa\_total*, *pontos\_vida\_total*, *ataque\_corpo*, *ataque\_distancia*, *dano\_maximo\_corpo* e *dano\_maximo\_distancia*.

O quadro 2 traz um exemplo de requisição sendo feita com os valores de entrada e a saída do resultado.



Quadro 2 - requisição feita aos personagens\_jogador e seu retorno

The screenshot shows a REST client interface with a POST request to `localhost:3000/api/v1/personagens_jogador/`. The request body is form-data with the following fields:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> nome	Samson 1515	
<input checked="" type="checkbox"/> nivel_personagem	15	
<input checked="" type="checkbox"/> defesa_total	22	
<input checked="" type="checkbox"/> pontos_vida_total	90	
<input checked="" type="checkbox"/> ataque_corpo	16	
<input checked="" type="checkbox"/> ataque_distancia	8	
<input checked="" type="checkbox"/> dano_maximo_corpo	15	
<input checked="" type="checkbox"/> dano_maximo_distancia	0	

The response is a JSON object:

```

1  {
2    "id": 15,
3    "nome": "Samson 1515",
4    "nivel_personagem": 15,
5    "defesa_total": 22,
6    "pontos_vida_total": 90,
7    "ataque_corpo": 16,
8    "ataque_distancia": 8,
9    "dano_maximo_corpo": 15,
10   "dano_maximo_distancia": 0
11  }

```

Fonte: autor

#### 4.1.3. Rota show

Se no *index* a rota devolve todos os objetos de um determinado *controller* existente, o *show* traz um objeto específico relacionado a *id* passada como parâmetro ao fim da URL, ou seja, se for fornecido o ID 14 para *personagens\_jogador*, a rota irá procurar pela linha 14 na tabela e retornará suas colunas, caso elas existam. Do contrário, emitirá um erro de valor não encontrado.

A tabela 12, a seguir, lista as rotas *show* de cada *controller*, junto com a *URL* interna e externa, o protocolo REST utilizado e sua ação.

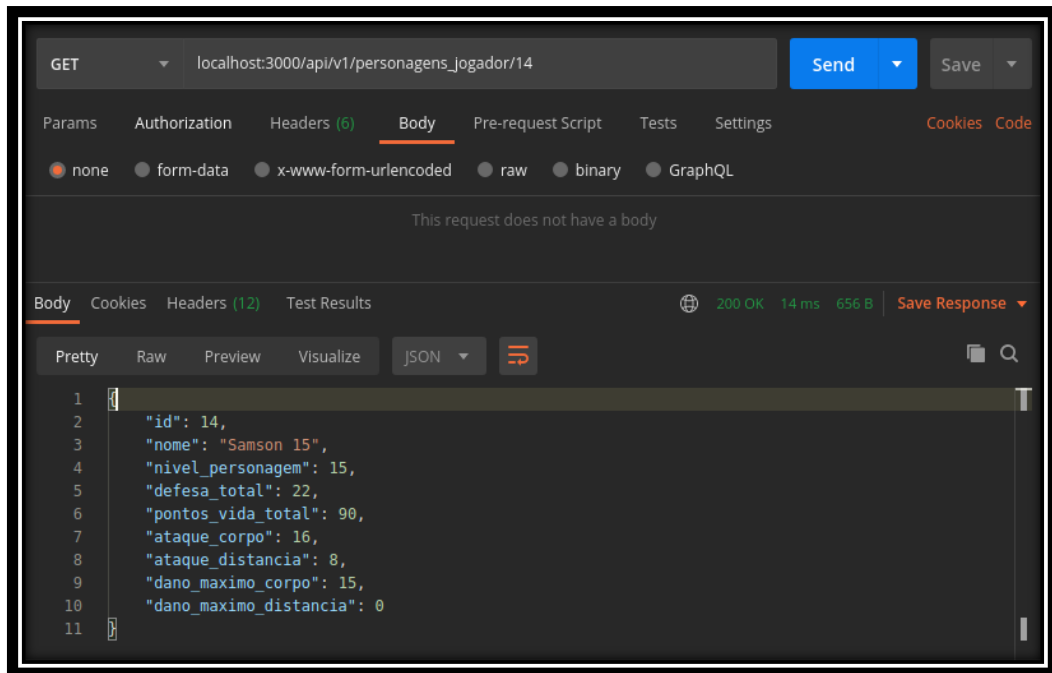
Tabela 12 - Descrição das rotas *Show*

URL interna	URL externa	Protocolo	Ação
api_v1_equipe_path	api/v1/equipes/:id(.:format)	GET	api/v1/equipes#show {:format=>:json}
api_v1_personagem_jogador_path	api/v1/personagens_jogador/:id(.:format)	GET	api/v1/personagens_jogador#show {:format=>:json}
api_v1_monstro_path	api/v1/monstros/:id(.:format)	GET	api/v1/monstros#show {:format=>:json}

Fonte: autor

Quando a rota termina de pegar o objeto requisitado, ela passa a chamada para a *view* de mesmo nome, *show.json.jbuilder*, que monta em formato *JavaScript Object Notation* (JSON) a saída dos dados. O quadro 3 traz um exemplo de requisição sendo feito com os valores de entrada e a saída do resultado.

Quadro 3 - requisição feita aos personagens\_jogador e seu retorno



Fonte: autor

#### 4.1.4. Rota update

O *update* é a rota que atualiza um dado existente. Essa rota poderá atualizar apenas um objeto por vez, caso em que será necessário informar qual o objeto que se pretende alterar, informando o ID no final da URL, e quais colunas serão atualizadas, informando os novos valores que irão substituí-las. Como resultado, a requisição irá redirecionar à rota *show*, mostrando o objeto atualizado ou, caso tenha ocorrido algum erro, mostrará a mensagem de erro na tela.

A tabela 13, a seguir, lista as rotas *update* de cada *controller*, juntamente com a URL interna e externa, o protocolo REST utilizado e sua ação.

Tabela 13 - Descrição das rotas Update

URL interna	URL externa	Protocolo	Ação
api_v1_equipe_path	api/v1/equipes/:id(.:format)	PATCH/P UT	api/v1/equipes#update {:format=>:json}
api_v1_personagem_jogador_path	api/v1/personagens_jogador/:id(.:format)	PATCH/P UT	api/v1/personagens_jogador#update {:format=>:json}

Fonte: autor

O quadro 4 a seguir traz um exemplo de requisição sendo feito com os valores de entrada e a saída do resultado.

Quadro 4 - requisição feita aos personagens\_jogador e seu retorno

The screenshot shows a REST client interface with the following details:

- Method:** PATCH
- URL:** localhost:3000/api/v1/personagens\_jogador/14
- Body (form-data):**

KEY	VALUE	DESCRIPTION
nome	Samson 1515	
nivel_personagem	15	
- Response:** 200 OK, 59 ms, 658 B
- Response Body (JSON):**

```

1  {
2    "id": 14,
3    "nome": "Samson 1515",
4    "nivel_personagem": 15,
5    "defesa_total": 22,
6    "pontos_vida_total": 90,
7    "ataque_corpo": 16,
8    "ataque_distancia": 8,
9    "dano_maximo_corpo": 15,
10   "dano_maximo_distancia": 0
11  }

```

Fonte: autor

#### 4.1.5. Rota *destroy*

A rota *destroy* é responsável por excluir uma linha de determinada tabela de acordo com o *controller*. A exclusão é unitária como o *update*. Por essa razão, será necessário passar o *id* no final da URL para determinar o objeto. Ao excluir, a requisição não retorna nada, mas se houver algum erro, será retornada uma mensagem de falha.

A tabela 14, a seguir, lista as rotas *delete* de cada *controller*, juntamente com a URL interna e externa, o protocolo REST utilizado e sua ação.

Tabela 14 - Descrição das rotas *Delete*

URL interna	URL externa	Protocolo	Ação
api_v1_equipe_path	api/v1/equipes/:id(.:format)	DELETE	api/v1/equipes#destroy{:format=>:json}
api_v1_personagem_jogador_path	api/v1/personagens_jogador/:id(.:format)	DELETE	api/v1/personagens_jogador#destroy{:format=>:json}

Fonte: autor

## 4.2. Algoritmo Genético

O algoritmo genético foi construído como um *interactor*, que é chamado toda vez que a ação *gerar\_monstro* for executada. Como o monstro é calculado em relação a um grupo de personagens jogadores, será necessário passar para a rota qual o *id* da equipe que este monstro fará parte, pois o monstro deverá ser relacionado com a equipe e nele será possível saber quais PJs estão associados.

Uma vez que a rota é invocada, será necessário passar um conjunto *hash* como parâmetro para o *interactor*, que deverá conter o tamanho da população, a taxa de mutação, a quantidade limite de iterações, o *id* da equipe e o nível de dificuldade que o monstro está sendo criado, podendo ser *minion*, *normal* ou *boss*. Para este trabalho, foram definidos os valores 400, 0.03, 50, 14 e *normal*, respectivamente, para os valores da *hash*.

Dentro do *interactor*, os parâmetros são distribuídos às suas respectivas variáveis e uma função auxiliar chamada `calcular_taxa_dificuldade` será invocada. Nessa função, será passado o nível estabelecido para o monstro e, de acordo com o nível do monstro, serão selecionadas as variáveis `const_defesa`, `const_dano`, `const_erro_atk` e `const_vida_maxima`. Esses valores são utilizados durante a função de avaliação e serão explicados no tópico 4.3. A tabela 15, a seguir, mostra quais os valores atribuídos de acordo com a dificuldade selecionada.

Tabela 15 - Relacionamento da dificuldade com as variáveis auxiliares

<b>Dificuldade</b>	<b>Constantes</b>
<i>Minion</i>	<code>const_defesa = 0,25</code> <code>const_dano = 4</code> <code>const_erro_atk = 0,5</code> <code>const_vida_maxima = 2</code>
Normal	<code>const_defesa = 0,5</code> <code>const_dano = 3</code> <code>const_erro_atk = 0,25</code> <code>const_vida_maxima = 3</code>
<i>Boss</i>	<code>const_defesa = 0,75</code> <code>const_dano = 2</code> <code>const_erro_atk = 0,1</code> <code>const_vida_maxima = 4</code>

Fonte: Autor

Além das variáveis, a subfunção `montar_array_pjs` é chamada, pegando o grupo de aventureiros e agrupando-os, por atributo, em listas distintas que serão utilizadas durante o algoritmo genético.

Com as variáveis necessárias atribuídas, o método `algoritmo_genetico` é chamado. Esse método recebe como parâmetros o tamanho da população, a taxa de mutação e a quantidade de iteração que o algoritmo genético, já definido anteriormente, irá utilizar.

Antes de iniciar o processo de evolução, será necessário criar a primeira geração. O cromossomo deste projeto contém seis genes representando as características do monstro, sendo elas respectivamente: defesa, pontos de vida máximo, ataque corpo a corpo, ataque à distância, dano corpo a corpo e dano à distância. O método pela geração dos indivíduos é o `gerar_cromossomo`, cujos genes recebem valores inteiros de maneira aleatória, conforme mostrado na tabela 16.

Tabela 16 - Período numérico sortido para cada gene do cromossomo

<b>Gene</b>	<b>Período de seleção</b>
1 – Defesa	10 a 40
2 – Pontos de vida máximo	10 a 250
3 – Ataque corpo a corpo	1 a 20
4 – Ataque à distância	1 a 20
5 – Dano corpo a corpo	1 a 25
6 – Dano à distância	1 a 25

Fonte: autor

Com a primeira geração criada, o processo evolutivo se inicia, chamando o método `avaliar_todos` que, ao passar a geração atual de cromossomos para ele, é calculado para cada indivíduo da geração, o seu grau de aptidão através da função `avaliacao` explicado no tópico seguinte. Após esta etapa, é verificado se algum cromossomo obteve o grau de aptidão ótimo e, caso tenha, o algoritmo termina, retornando este indivíduo ideal.

Caso não tenha encontrado o melhor indivíduo, o código continua passando para os operadores genéticos: `selecao_roleta`, `crossover` e `mutar_todos`, nesta ordem. Com os operadores rodados, é verificado nesta geração, se algum indivíduo sofreu mutação através do método `mutar_todos`. Por fim, os cromossomos são novamente reavaliados e, caso não encontre nenhum indivíduo perfeito, o ciclo se repete, passando para a próxima geração até um ser contemplado ou a quantidade de iterações definida pelo parâmetro ser atingido.

### 4.3. Função de Avaliação

Para o cálculo da aptidão do cromossomo, foi necessário dividir a função em quatro partes, pois os genes não apresentavam características em comum, dificultando a avaliação do indivíduo como um todo. Durante a primeira parte, foi calculado o primeiro gene, referente à defesa. Na segunda parte, o segundo gene, que representa a vida. Na penúltima parte, envolvendo o terceiro e quarto genes na verificação dos ataques. Por último, foram calculados os quinto e sexto genes, envolvendo o dano.

#### 4.3.1. Avaliação da defesa

Para a primeira avaliação, foi necessário calcular qual a defesa ideal este monstro precisaria ter para o grupo de aventureiro em estudo. Para isso, foi pego o jogador com o maior ataque, independentemente de ser corpo a corpo ou à distância, e foi somada a ele uma taxa

constante. Esta taxa, que representa a chance de desvio do monstro, é calculada multiplicando 20 (valor máximo num dado de 20 faces, utilizado durante um ataque) com a chance de desvio que o monstro apresenta, no caso de dificuldade normal, 50%. O resultado desta multiplicação é o valor mínimo no dado que o personagem com o ataque mais forte precisa tirar para acertar o monstro. Seguindo ainda o exemplo acima, supondo que o personagem mais forte tenha um modificador de 10, com a taxa também em 10 (metade de 20), a defesa ideal para o monstro será de 20.

Com a defesa ideal obtida, é realizada a subtração da defesa que o cromossomo tem e, em seguida, é feita a divisão de um pelo seu valor absoluto e, por fim, multiplicada por 100. Com a chance de obter uma divisão por zero, a verificação da subtração ser zero é realizada, sendo esta considerada como 1. A divisão por um é realizada porque é preciso inverter a proporção obtida, uma vez que, quanto mais próximo de zero ela for, melhor será a defesa do indivíduo, pois esse é o valor que mais se aproxima da defesa ideal. A multiplicação por 100 é realizada apenas para aumentar os valores divididos, uma vez que varia de 0,1 a 1. A tabela 17, a seguir, exemplifica essa situação, mantendo a defesa ideal como 20.

Tabela 17 - Exemplo de defesas de cromossomos e seus respectivos resultados

<b>Defesa do cromossomo</b>	<b>Equação</b>	<b>Resultado da equação apresentada</b>
20 ou 19	$  1/(20-19)  *100$	100
18	$  1/(20-18)  *100$	50
17	$  1/(20-17)  *100$	33,33
10	$  1/(20-10)  *100$	10
5	$  1/(20-5)  *100$	6,66

Fonte: autor

A partir da análise da tabela anterior, percebe-se que os resultados desta equação caem abruptamente para cada passo de distância da solução ideal e isso pode gerar um grande problema chamado de solução local. Para evitar isso, foi atribuído um bônus ao resultado da equação. Caso ela seja maior que 20 e menor que 100, o resultado deverá ser dobrado. Com isso, serão obtidos os valores mais próximos da solução ideal influenciados, deixando-os mais evidentes.



#### 4.3.2. Avaliação da vida

Para esta avaliação, será preciso calcular a soma do dano ponderado da equipe. Para isso, o dano do ataque mais forte de cada um é considerado, dividindo cada dano pela defesa ideal calculado anteriormente. Com esta soma, multiplicamos o resultado pela quantidade de turnos que o monstro idealmente precisa sobreviver, obtendo a vida ideal.

Com a vida ideal calculada, o gene de vida do monstro é subtraído da vida ideal e, em seguida, é dividido 1 pelo seu valor absoluto e multiplicado por 100. Além do problema da divisão por 0, se a vida do inimigo está diferente da ideal, por mais ou menos 5, a diferença é desprezível e, por isso, qualquer valor de 0 a 5 é considerado como 1. Além desta consideração, é atribuído um bônus para qualquer valor entre 99 e 20 do resultado, multiplicando estes valores por 2.

#### 4.3.3. Avaliação dos ataques

Alguns cálculos devem ser feitos antes de ser gerada a fórmula de avaliação. Primeiramente, é adotado o maior ataque que o cromossomo possui. Em seguida, é selecionada a maior defesa entre os PJs envolvidos e, por último, a taxa de desvio deste mesmo PJ é calculada.

A taxa é calculada multiplicando o valor 20 (maior resultado possível num dado de 20 faces) com a constante de erro que, por sua vez, é definida pela dificuldade, no caso do normal, 0.25. Isso quer dizer que com qualquer valor acima de 5 tirado pelo monstro no d20, ele deve conseguir acertar o aventureiro com maior defesa.

Com todas as variáveis calculadas, a seguinte fórmula será aplicada: somando o maior ataque da criatura com a taxa de desvio, do resultado dessa operação será subtraída a maior defesa do grupo. Em seguida, é dividido um pelo valor absoluto do cálculo anterior e, depois, multiplicado por 100. Vale lembrar que, pela possibilidade de surgir uma divisão por zero, caso a subtração seja zero, o valor um é atribuído.

#### 4.3.4. Avaliação dos danos

Para esta avaliação, serão necessários o dano do maior ataque que a criatura tem, a vida máxima do PJ com o maior valor, a constante dano que representa em quantos turnos o monstro deve idealmente drenar esta vida máxima a zero e a constante de acerto do monstro

que é obtido subtraindo de um a constante de erro dos ataques. Todas essas constantes são definidas pela dificuldade inicialmente estipulada.

Com as variáveis calculadas, a vida máxima do maior PJ é dividida pela constante de dano e seu resultado, multiplicado pela constante de acerto. Com isso, uma taxa de dano ideal é obtida e seu valor, subtraído do dano que o monstro possui com seu melhor ataque. É dividido um pelo resultado absoluto desta subtração e, em seguida, multiplicado por 100. Nesse cálculo, deve ser observado que a subtração pode resultar em zero, gerando uma divisão por zero e, portanto, caso ocorra, o valor um é considerado.

Após o cálculo dessas quatro avaliações, uma média é feita entre elas e o resultado é armazenado como grau de aptidão do indivíduo.

## 5. RESULTADOS

Antes de discutir os resultados obtidos por este projeto, é necessário frisar que o algoritmo genético é um código estocástico e que, por essa razão, pode apresentar variados resultados.

Para a geração dos dados, foram montadas três equipes com quatro PJs, sendo a primeira no nível um, a segunda nível cinco e, por fim, a última no nível quinze. A tabela 18 mostra as estatísticas da equipe de nível 5, enquanto a tabela 19 mostra a equipe de nível 1 e, por fim, a tabela 20 apresenta o grupo de nível 15.

Tabela 18 - Equipe I com jogadores de nível 5

<b>Nome</b>	<b>Vida máxima</b>	<b>Ataque corpo a corpo</b>	<b>Ataque à distância</b>	<b>Dano corpo a corpo</b>	<b>Dano à distância</b>	<b>Defesa</b>
Samson Sansone	78	7	3	11	0	20
Doutor Joseph	29	2	10	2	22	11
Rakin	42	1	9	3	19	17
Kahyn	37	3	8	7	12	15

Fonte: autor

Tabela 19 - Equipe II com jogadores de nível 1

<b>Nome</b>	<b>Vida máxima</b>	<b>Ataque corpo a corpo</b>	<b>Ataque à distância</b>	<b>Dano corpo a corpo</b>	<b>Dano à distância</b>	<b>Defesa</b>
Assis	24	6	1	16	0	15
Edgy	13	0	6	0	12	16
Caça Dor	16	0	6	0	20	16
Ingran	28	6	0	12	0	15

Fonte: autor

Tabela 20 - Equipe III com jogadores de nível 15

<b>Nome</b>	<b>Vida máxima</b>	<b>Ataque corpo a corpo</b>	<b>Ataque à distância</b>	<b>Dano corpo a corpo</b>	<b>Dano à distância</b>	<b>Defesa</b>
Rakin 15	42	6	18	3	19	19
Joseph 15	69	7	19	4	22	13
Kahyn 15	37	8	7	17	7	17
Samson 15	90	16	8	15	0	22

Fonte: autor

Outros dados importantes são os parâmetros definidos para a obtenção destes resultados, que foram: tamanho da população, estipulado em 400, a taxa de mutação em 3%, quantidade de iteração por geração em 50 e a taxa de dificuldade como normal. Com os valores definidos, o programa rodou duas iterações de 100 evoluções para cada equipe, totalizando 600 iterações.

Para o grupo de nível 1, foi obtida uma taxa de acerto em 74% e 80% das vezes rodadas, ou seja, na primeira vez, com 100 iterações, 74 vezes o indivíduo ideal foi encontrado e, na segunda vez, 80 de 100 iterações retornaram o resultado esperado. Já para a equipe de nível 5, foi obtida uma porcentagem de 81% e 80% de acerto e, por fim, para o grupo nível 15, foi obtida 85% e 90%. A tabela 21, a seguir, traz exemplos de cromossomos ideais gerados pela equipe I com os PJs no nível 5 e as tabelas 22 e 23 trazem, respectivamente, exemplos de cromossomos ideais gerados pela equipe II, com PJs de níveis 1 e equipe III, com PJs de nível 15.

Tabela 21 - Exemplos de cromossomos da Equipe I de nível 5

<b>Equipe I - Iteração 1</b>	
1	[20, 84, 9, 15, 2, 19]
2	[19, 87, 16, 12, 20, 3]
3	[21, 89, 6, 15, 4, 19]
4	[19, 85, 6, 16, 14, 19]
5	[20, 81, 12, 14, 24, 20]
<b>Equipe I - Iteração 2</b>	
1	[18, 81, 13, 14, 19, 19]
2	[22, 86, 14, 2, 20, 3]
3	[20, 82, 1, 16, 5, 20]
4	[22, 86, 16, 8, 20, 18]
5	[21, 85, 15, 16, 19, 20]

Fonte: autor

Tabela 22 - Exemplos de cromossomos da Equipe II de nível 1

<b>Equipe II - Iteração 1</b>	
1	[15, 71, 12, 10, 6, 9]
2	[16, 65, 12, 9, 7, 6]
3	[15, 65, 11, 2, 6, 8]
4	[16, 71, 12, 3, 7, 21]
5	[15, 63, 5, 10, 5, 6]
<b>Equipe II - Iteração 2</b>	
1	[16, 63, 10, 7, 7, 1]
2	[17, 67, 12, 10, 7, 15]
3	[18, 63, 12, 2, 6, 13]
4	[15, 64, 9, 11, 1, 6]
5	[14, 64, 3, 10, 7, 7]

Fonte: autor

Tabela 23 - Exemplos de cromossomos da Equipe III de nível 15

<b>Equipe III - Iteração 1</b>	
1	[27, 135, 14, 17, 22, 23]
2	[27, 133, 16, 2, 22, 5]
3	[30, 136, 16, 10, 22, 19]
4	[29, 132, 16, 10, 22, 14]
5	[28, 136, 16, 7, 22, 15]
<b>Equipe III - Iteração 2</b>	
1	[31, 139, 1, 16, 5, 23]
2	[29, 132, 1, 16, 10, 23]
3	[30, 133, 14, 17, 15, 23]
4	[27, 137, 7, 16, 22, 23]
5	[28, 134, 11, 18, 3, 22]

Fonte: autor

A título de medida comparativa, foi selecionado um monstro do livro base para cada nível de equipe testada, ou seja, um monstro de ND 1 para a equipe nível 1, um de ND 5 para a equipe nível 5 e assim, sucessivamente.

O monstro de ND 1 selecionado foi o Gorlogg (Tormenta20, pg 276). Esta criatura tem uma defesa de 16, 36 pontos de vida, uma mordida com 11 de ataque e o dano em  $1d8 + 5$  (significa que é rolado um dado de 8 faces e adicionado 5 ao resultado).

Se o Gorlogg for comparado com os cromossomos obtidos na equipe II, que varia sua defesa entre 15 e 16, em sua maioria, conclui-se que o inimigo genérico também está com a defesa ideal, pois como discutido na sessão 4.3.1, o cálculo da defesa ideal se dá pelo ataque mais forte da equipe somado à taxa de acerto, que está em 10, ou seja 16.

Quando a vida é comparada, é perceptível que o Gorlogg tem 36 pontos de vida, enquanto a média dos cromossomos está em 65, ou seja, quase o dobro. Isso significa que o monstro básico é fraco para esta equipe, pois pelos cálculos apresentados na sessão 4.3.2, o inimigo ideal é aquele que consegue sobreviver por 3 turnos (se considerada a dificuldade normal) utilizando o dano ponderado da equipe. Isso equivale a afirmar que para o monstro cumprir com estes requisitos, deverá ter, aproximadamente, 67,5 pontos de vida, pois a equipe conseguirá causar 22,5 pontos de dano em uma rodada.

O ataque do monstro trazido pelo livro está próximo do obtido pelos cromossomos (desde que considerados apenas o melhor ataque corpo a corpo e à distância), ou seja, esses

dois inimigos estão equilibrados para a equipe II, pois apresentam uma média de 75% de acerto contra o personagem com a defesa mais alta.

Como no livro base o dano da criatura é variável, deverá ser considerada a metade do valor máximo obtida pelo dado adicionada ao dano fixo, ou seja, 9. Conclui-se que o monstro genérico está um pouco acima do ideal, pois pelos cálculos da sessão 4.3.4, o dano ideal para esta equipe seria em 7.

Para a equipe I, o monstro selecionado foi o Wyvern, uma criatura reptiliana alada com pouca inteligência e sem poderes mágicos, também retirada do capítulo de ameaças do livro básico. Ela apresenta 21 de defesa, 144 pontos de vida, dois ataques com 17 em acerto e dois danos um sendo  $2d6+7$  (a soma de dois dados de 6 faces mais 7) e o outro  $1d6+7$ .

Enquanto a defesa entre os cromossomos e o Wyvern estão próximas, a vida novamente apresenta discrepância, uma vez que os cromossomos apresentarão uma média de 65 pontos de vida, e o Wyvern com 144, ou seja, quase o dobro. Isto ocorre porque a criatura genérica precisa tentar considerar a possibilidade de receber dano de outras fontes além do ataque comum, seja por magias ou habilidades e, neste nível, a maioria das classes jogáveis ganham um aumento significativo em poder, a exemplo da liberação de magias mais fortes para os arcanistas ou da possibilidade do guerreiro passar a poder fazer dois ataques por turno. Entretanto, os cromossomos não preveem esta situação, seguindo apenas os cálculos já apresentados neste projeto.

O modificador de ataque também está próximo entre o Wyvern e os cromossomos, mas os cromossomos atacam apenas uma vez por turno enquanto o Wyvern realiza dois ataques. Mesmo que o dano desses ataques seja somado, os valores variáveis, considerando a metade do valor máximo, novamente estarão próximos do dano do cromossomo. Nesse caso, o monstro genérico terá duas chances de tentar acertar o grupo inimigo, tentando garantir que ao menos um dos ataques consiga causar um pouco do dano, pois enquanto ele tem apenas um turno, o grupo inimigo, no geral, possui quatro personagens, cada um com um turno, tentando acertá-lo.

Por fim, para a equipe nível 15, foi selecionado o Dragão Venerável para comparação. Esse inimigo possui uma defesa de 43, 441 pontos de vida e realiza 3 ataques com modificador de acerto em 33 e um dano com  $4d6+17$  e dois com  $3d6+17$ . Em comparação com os dados exibidos para a equipe III, é perceptível a diferença em todas as características, uma vez que o dragão apresentará valores maiores que qualquer cromossomo gerado. O motivo segue a mesma lógica do Wyvern anteriormente citado. O dragão precisa sobreviver a 4 personagens nível 15 com suas habilidades heroicas. Nesta fase, o arcanista consegue alterar a realidade, causar 80

pontos de dano em um turno ao passo em que o guerreiro fará vários ataques, reduzindo à metade qualquer dano tomado.

A partir dessas comparações, é possível perceber que nos níveis iniciais, em que as habilidades ainda não escalaram tanto, o cromossomo e o monstro genérico se aproximam muito, mas conforme os níveis vão aumentando, as discrepâncias surgem cada vez mais por influência de magias, habilidades e até itens.

É importante ressaltar que, para medida de comparação, foi utilizada apenas as estatísticas aqui mencionadas, não sendo consideradas as habilidades trazidas no livro para estas criaturas, como o agarrar aprimorado do GORLOGG ou o veneno da cauda do WYVERN, ou as magias que o dragão venerável consegue lançar.

Como último teste, foi feita uma simulação simples de combate com três cromossomos da equipe I. Para calcular a ordem do turno, foi rolado um d20 para cada participante e ordenada em ordem decrescente. Como não estão sendo consideradas habilidades nem movimentação, cada turno terá apenas um teste de ataque e, em caso de acerto, causará dano.

A primeira simulação foi com o cromossomo 5 da primeira iteração e a ordem da iniciativa (ordem dos turnos) foi: Kayn, cromossomo, Samson, Rakin e Joseph.

No primeiro turno, Kayn não conseguiu acertar o inimigo, pois acabou tirando 4 no dado, totalizando 12 no ataque. O cromossomo acerta o Samson porque é o personagem com maior defesa, causando 19 de dano, deixando Samson com 59 de vida. Samson, no seu turno, acabou errando o monstro com um resultado 16 no acerto. Rakin também errou, tirando 17 no total e, por último, Joseph acertou o cromossomo com 27 no resultado, causando 22 pontos de dano, deixando o cromossomo com 67 pontos de vida.

O segundo turno teve início com Kayn acertando o cromossomo, causando 15 de dano e deixando o cromossomo com 52 pontos de vida. O cromossomo seguiu acertando Samson, causando mais 19 pontos de dano, deixando-o com 40 pontos de vida. Samson devolveu o ataque, acertando a criatura e causando 11 de dano, deixando o indivíduo com 41 pontos. Em seguida, Rakin e Joseph erraram o inimigo.

No último turno, Kayn acabou errando o cromossomo, mas Samson acertou, deixando-o com 21 pontos de vida. Na vez de Samson, ele errou o ataque, mas Rakin e Joseph conseguiram acertá-lo, eliminando o cromossomo.

No segundo combate, o cromossomo 4 da iteração 2 foi selecionado e a iniciativa definida como: Rakin, Samson, Cromossomo, Kayn e Joseph por último.

Rakin inicia seu turno errando a criatura juntamente com Samson, mas o cromossomo consegue acertar, causando 20 pontos de dano em Samson, deixando-o com 58 pontos de vida.



Kayn e Joseph acertam o monstro, causando 15 e 22 pontos de dano, respectivamente, deixando, ao final, o cromossomo com 49 pontos de vida.

No segundo turno, Rakin acerta, causando 19 pontos de dano, deixando o cromossomo com 30 pontos de vida. Samson também acerta, causando 11 pontos de dano, diminuindo a vida do inimigo para 19 pontos. O cromossomo devolve o ataque, acertando Samson e deixando-o com 38 pontos de vida. Kayn e Joseph erram os próximos dois ataques. A criatura é finalizada por Rakin no começo da terceira rodada.

O último combate foi contra o cromossomo 5 da primeira iteração, com a iniciativa estabelecida em: Samson, Joseph, Cromossomo, Rakin e Kayn.

No primeiro turno, Samson começa errando o inimigo, mas Joseph acerta, causando 22 pontos de dano, deixando o cromossomo com 59 pontos de vida. O inimigo devolve o acerto, causando danos a Samson e deixando-o com 58 pontos de vida. Rakin e Kayn acertam o cromossomo, deixando-o com 25 pontos de vida. No segundo turno, apenas Joseph e o cromossomo acertam, deixando o monstro com 3 de vida e Samson, com 38. No fim, Samson finaliza a criatura no início do terceiro turno.

Após as três simulações, conforme proposto pelo presente trabalho, foi analisado que o cromossomo sobreviveu por três turnos, acertou quase todas as vezes que atacou e conseguiu desviar da maioria dos golpes. Verificou-se, entretanto, que não foi impossível atingir o cromossomo. Por outro lado, embora o guerreiro tenha sobrevivido, sua vida, ao final, estava abaixo dos 50%. Os cromossomos estão, portanto, seguindo o *flow*, o que faz com que os inimigos sejam considerados ideias.

## 6. CONSIDERAÇÕES FINAIS

O presente projeto propôs o desenvolvimento de um *webservice* capaz de gerar um monstro ideal a um grupo de aventureiros a partir de um algoritmo genético com o auxílio da teoria do *flow*. O objetivo geral foi alcançado, sendo possível encontrar um indivíduo adaptado o suficiente que representasse o monstro ideal para um grupo específico de PJs. Para que isso fosse possível, o primeiro passo foi satisfazer os objetivos específicos, tais como o desenvolvimento da API que serviu de sustento para o algoritmo genético que, nada mais é que o próprio algoritmo que gerou o inimigo ideal. Nesse contexto, a teoria do *flow* foi indispensável no que tange à definição do monstro ideal para o grupo alvo.

A função de avaliação, método responsável pela avaliação dos cromossomos, embora não tenha sido perfeita, apresentou uma taxa de acerto suficiente, deixando espaço para melhoria e aperfeiçoamento. Outro fator passível de análise é a simplicidade das propriedades consideradas para a criação do monstro.

Os atributos gerados para o monstro, assim como as habilidades consideradas dos PJs, são apenas os valores bases de suas fichas, sendo que não foram consideradas nenhuma habilidade ou magia, o que para situações reais, torna-se inviável, pois se considerarmos um conjurador apenas pelos seus atributos físicos, ele será considerado fraco e agregará pouco valor ao grupo. Porém, a partir do momento em que as magias forem consideradas, o conjurador poderá facilmente se tornar no aventureiro com maior dano da equipe. Outro exemplo é o do ladino do grupo, que depende muito de suas habilidades de classe para conseguir causar dano, tal como a habilidade ataque furtivo, que pode facilmente triplicar seu dano se usada corretamente. O mesmo pode ser dito para a criatura gerada pelo projeto.

Um monstro pode ser perigoso apenas por suas características brutas, como quantidade de vida ou dano por ataque infligido, mas um inimigo que apresente apenas essas características será considerado chato e monótono, uma vez que não apresenta diversidade. A partir da adição de habilidades e poderes únicos à criatura, o monstro é moldado com mecânicas distintas, por exemplo, um Troll da floresta e um dragão vermelho têm sua vida máxima alta e um bom valor de dano por turno. Porém, o que os tornará únicos será que o Troll, além de ser uma criatura vegetal e fraca contra o fogo, recebendo dano dobrado dessa fonte, possui a habilidade de se regenerar todo turno, recuperando parte da vida perdida. O dragão vermelho, por sua vez, tem o famoso sopro, é imune a qualquer habilidade de fogo e realiza três ataques em um único turno.

Pelo fato do *software* desenvolvido ser uma API, seus resultados são apresentados de maneira mais técnica, através do JSON, o que não é considerado muito amigável pelos usuários,

o que faz com que o desenvolvimento de um aplicativo *Front End* consiga consumir as rotas da API, apresentando resultados mais amigáveis.

Considerando os pontos analisados até aqui, faz-se necessário elencar, a título de sugestão para os trabalhos futuros, a adição de habilidades e magias tanto no grupo de PJs, tornando a simulação mais próxima da realidade, quanto na criação dos monstros, adicionando unicidade e complexidade às suas mecânicas, além da criação de um aplicativo *Front End* que possa receber os dados da API, mostrando-se mais amigável para o usuário.

## REFERÊNCIAS

- BERNACCHIA, M., Hoshino, **AI platform for supporting believable combat in role-playing games**. In: Proceedings of the 19th Game Programming Workshop in Japão, 2014. p. 139–144
- BRAUNER, G. et al. **Tormenta RPG**. Edição Revisada. Porto Alegre: Editora Jambo, 2013.
- COSTA, L. **Algoritmos Evolucionários em Otimização Uni e Multi-Objectivo**. Tese de Doutorado, Ramo de Engenharia de Produção e Sistemas, Braga: Universidade do Minho, 2003
- COSTA, Wagner R. **Investigando a conversão da escrita natural para registros em escrita algébrica em problemas envolvendo equações de primeiro grau**. F. Dissertação de Mestrado – Educação Matemática e Tecnológica, Universidade Federal de Pernambuco, Pernambuco, 2010.
- CORTE, Felipe Della. **Caverna do saber: um bom desafio como encontrar o equilíbrio de ND ideal para o seu grupo em T20**. Dragão Brasil, Porto Alegre, ed. 162, p. 41-44, 2020
- CHAGAS, E. M. P. F. et al. **Método de Segmentação de Objetos em Imagens Baseado em Contornos Activos e Algoritmo Genético**. In: Congresso de Métodos Numéricos em Ingeniería, 1., 2009, Barcelona. Anais... Barcelona: Semni, 2009. P. 491-507.
- CHRISTINA, Camila. **TOROG: Aplicação para Suporte ao Tormenta RPG na Tomada de Decisão dos Oponentes Utilizando Algoritmo Genético**. f. Trabalho de Conclusão de Curso (Especialização) – Faculdade de Engenharia de Computação, Pontifícia Universidade Católica de Goiás, Goiás, 2017.
- CSIKSZENTMIHALYI, M. **Flow, the secret to happiness**. 2004. Disponível em: <[http://www.ted.com/talks/mihaly\\_csikszentmihalyi\\_on\\_flow](http://www.ted.com/talks/mihaly_csikszentmihalyi_on_flow)>. Acesso em: 14 maio 2021.
- FILHO, Paulo A. C.; POPPI, Ronei Jesus. **Algoritmo genético em química**. Química Nova, São Paulo, v. 22, n.3, p. 405-411, Junho 1998. Disponível em: <[https://www.scielo.br/scielo.php?pid=S0100-40421999000300019&script=sci\\_arttext](https://www.scielo.br/scielo.php?pid=S0100-40421999000300019&script=sci_arttext)>. Acessado em: 17 maio 2021.
- FREITAS, Marcos A. **Equação do 1º grau: métodos de resolução e análise de erros no ensino médio**. F. Dissertação de Mestrado – Educação Matemática, Pontifícia Universidade Católica de São Paulo, São Paulo, 2002
- GUIMARÃES, Thiago S. M. **Apoio à Síntese de Modelos Estruturais de Software Orientado a Objetos Utilizando Algoritmos Genéticos Co-Evolucionários**. f. Dissertação de Mestrado – Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2003.
- ISHIVATARI, Luiz H. U. et al. **Algoritmos genéticos com função de avaliação dinâmica para o problema de predição de estruturas de proteínas**. In Anais do X cong. Brasileiro de Inteligência Computacional in Ceará, 2011.

KIILI, K. et al. **The design Principles for Flow Experience in Educational Games**. In Procedia Computer Science, Finland. vol. 15 p. 78-91, 2012.

LINDEN, Ricardo. **Algoritmos Genéticos**. Rio de Janeiro: Ed. Ciência Moderna Ltda, 2012.

LOPEZ, Shane J. **The Oxford Handbook of Positive psychology**. New York: OXFORD university press, 2009.

LUCAS, Diogo C. **Algoritmos Genéticos: uma Introdução**. Apostila – Universidade Federal do Rio Grande do Sul, Rio Grande do Sul 2002

MALAQUIAS, Neli Gomes Lisboa. **Uso dos Algoritmos Genéticos para a Otimização de Rotas de Distribuição**. Dissertação (Mestrado em Ciências) – Universidade Federal de Uberlândia, Uberlândia, 2006

MASSARELLA, Fábio Luiz. **Motivação Intrínseca e o Estado Mental Flow em corredores de Rua**. 2008. f. Dissertação de Mestrado (Pós-Graduação) – Universidade Estadual de Campinas, São Paulo. 2008.

MEARLS, M.; CRAWFORD, J. **Player's HandBook**. USA: Wizards of the Coast LLC, 2014.

MEARLS, M.; CRAWFORD, J. **Dungeon Master's Guide**. USA: Wizards of the Coast LLC, 2014.

NETO, Sílvio P. **Computação Evolutiva: desvendando os algoritmos genéticos**. f. Dissertação de Mestrado – Faculdade de Jaguariúna-FAJ, São Paulo, 2011.

OBADÃ, Daniel R. **Flow Theory and Online Marketing Outcomes: A Critical Literature Review**. In Procedia Economics and Finance, Romania. vol. 6, p. 550-561, 2013

RYE, Connie. et al. **Biology**. Texas: OpenStax, 2016.

SILVA, T. et al. **A Teoria do Flow na contribuição do engajamento estudantil para apoiar a escolha de jogos no ensino de programação**. Pernambuco: SBIE, 2015.

SVALDI, Guilherme Dei. et. al. **Tormenta20**. Porto Alegre: jambo, 2020

SVALDI, Guilherme Dei. **Caverna do saber: lidando com ameaças**. Dragão Brasil, Porto Alegre, ed. 157, p. 45-48, 2020

PATIL, S.; Bhende, M. (2014). **Comparison and analysis of different mutation strategies to improve the performance of genetic algorithm**. In: International Journal of Computer Science and Information Technologies in 2014 vol.5(3).

WAZLAWICK, Raul Sidnei. **Metodologia de Pesquisa para Ciência da Computação**. 2ª Ed. Rio de Janeiro: Elsevier, 2014.

## GLOSSÁRIO

**Ataque furtivo:** Uma das principais habilidades de causar dano do ladino. Representa sua capacidade de acertar onde dói, seja um ponto fraco do inimigo, uma oportunidade surgida do descuido do inimigo ou por estar furtivo e atacar surpreendendo o inimigo.

**Atletismo:** É uma das perícias do aventureiro e representa sua capacidade atlética, como escalar montanhas, cruzar rios a nado e saltar sobre lava.

**Bola de fogo:** Uma das magias mais clássicas do arcanista, esta magia cria uma bola de fogo de 6m de raio que explode onde o conjurador apontar, explodindo tudo e todos que estiver no espaço.

**Mana (ou Pontos de Mana):** Diferente dos clássicos RPG em que apenas aquele que mexe com o arcano tem a mana, no T20 todos possuem essa essência. Representa a energia, determinação e força interior do personagem e é o que permite usar as diversas habilidades de classe, inclusive magias.

**Personagem do Mestre:** o Personagem do Mestre representa qualquer figura na aventura que o controle não fica com os jogadores, e sim com o mestre. Seja o velhinho que sempre pede socorro na taberna, o Rei da capital ou o monstro temível que está atacando a vila.

**Perícia:** São habilidades mundanas que um personagem pode possuir. Essas habilidades costumam ser a maneira que ele irá enfrentar o mundo. Por exemplo, Diplomacia é uma perícia e representa a eloquência do personagem, sua capacidade de fala, convencimento e negociação, enquanto Furtividade é a capacidade do jogador se esgueirar sem ser notado. Cada jogador começa com x perícias treinadas que geralmente são ditadas pela classe escolhida, mas pode vir de outras fontes. Como na realidade uma pessoa consegue ser melhor em certos aspectos que outros, através de treino, talento ou até sorte. Por isso, ao selecionar uma perícia treinada significa que seu aventureiro levou tempo para aperfeiçoar aquela técnica.

**Reflexos:** Uma perícia que representa sua reação e gingado. Mede sua capacidade de reagir à ameaças como uma armadilha se ativando no seu pé ou sair da frente a tempo da bola de fogo que o mago do grupo acabou de lançar em sua direção.

## APÊNDICE

Diagrama Entidade relacionamento do Banco de dados

