**CENTRO DE EDUCAÇÃO PROFISSIONAL HERMANN HERING**

**CURSO TÉCNICO EM INFORMÁTICA**

**PROTOTIPO DE SISTEMA**

**AUTOCOMPLETE**

**GUSTAVO HENRIQUE SPIESS**

**LUCAS GABRIEL DA COSTA**

**Trabalho de Conclusão de Curso**

**Blumenau - SC**

**2016**

**GUSTAVO HENRIQUE SPIESS**

**LUCAS GABRIEL DA COSTA**

**PROTOTIPO DE SISTEMA**

**AUTOCOMPLETE**

Trabalho de Conclusão de Curso feito com o âmbito de efetuar uma pesquisa em torno do paradigma de Inteligência Artificial focando em Software evolutivo.

Orientador: Prof. Fabio Busnardo, Esp.

Blumenau - SC

2016

**GUSTAVO HENRIQUE SPIESS**

**LUCAS GABRIEL DA COSTA**

**PROTÓTIPO DE SISTEMA DE AUTOCOMPLETE**

Trabalho de Conclusão de Curso aprovado para obtenção do grau de Técnico em Informática do Curso Técnico Integrado ao Ensino Médio em Informática – CEDUP Blumenau, pela banca examinadora formada por:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Presidente: Prof.º Fábio Busnardo, Esp.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Membro: Prof.º Jefferson Gregori, Esp.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Membro: Prof.ª Cláudia Neli de Souza Zambon, Mst.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Orientador TCC: Prof.º Fábio Busnardo, Esp.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Coordenador de Curso: Prof.º Vilmar Dias, Esp.

Blumenau -SC

2016

Dedicamos esse trabalho a todos os professores, que durante os últimos quatro anos nos orientaram e guiaram em rumo ao conhecimento.

**AGRADECIMENTOS**

À Fabio Busnardo, pelos conselhos sempre úteis e precisos com que, sabiamente, conduziu este trabalho.

TODO: CAPAR FORA SE NÃO HAVER RELAÇÃO COM O TRAMPO , OU ARRUMAR UMA NOVA

“Raramente estou mais feliz do que quando passo um dia inteiro a programar o meu computador para fazer automaticamente uma tarefa que de outra forma demoraria uns bons dez segundos a fazer à mão. ”

- Douglas Adams -

RESUMO

Esse trabalho é a exploração da possibilidade do uso dos conceitos de software evolutivo na construção de um autocomplete, com a descrição da construção de uma biblioteca genérica para tal metodologia, de forma que pode ser reutilizada posteriormente. O foco não é propriamente a construção de uma aplicação, mas a descrição de uma metodologia, suas características positivas, negativas e seus limites. Os esforços, apesar de não terem conseguido atingir as soluções mais otimizadas tiveram sucesso em demonstrar a execução de dois tipos diferentes de análises na aplicação do jovem campo dos algoritmos evolutivos, como solução de problemas e método de busca.

ABSTRACT

This work is the exploring the possibility of using software evolutionary concepts in building an autocomplete with the description of the general construction of a library for this method, so that it can be reused afterwards. The focus is not exactly the construction of an application, but the description of a methodology, its positive features, negative and its limits. The efforts, although failed to achieve the most optimized solutions have succeeded in demonstrating the implementation of two different types of analysis in the application of the young field of evolutionary algorithms, such as problem solving and search method.

**Palavras-chave:** Computação evolutiva. Software evolutivo. Algoritmo evolutivo. Autocomplete. Caixeiro-viajante.

INDICE DE IMAGENS

[Exemplo de Tag de HTML 25](#_Toc468315028)

[Diagrama de Caso de Uso 32](#_Toc468315029)

[Diagrama de Atividade - Apresenta Sugestões 33](#_Toc468315030)

[Diagrama de Atividade - Gerenciador Distribui texto 34](#_Toc468315031)

[Diagrama de Atividade - Gerenciador 34](#_Toc468315032)

[Diagrama de Atividade - Gerenciador Elimina Sugestores. 35](#_Toc468315033)

[Diagrama de Atividade - Gerenciador reproduz entidades. 36](#_Toc468315034)

[GerenciadorMatrix 36](#_Toc468315035)

[Sugestor 37](#_Toc468315036)

[MER do Banco de Dados (parte 1) 38](#_Toc468315037)

[MER do Banco de Dados (parte 2) 38](#_Toc468315038)

[Diagrama de Classes - Caixeiro Viajante 41](#_Toc468315039)

[Interface GerenciadorEvo 42](#_Toc468315040)

[Interface EntidadeEvo 42](#_Toc468315041)

[Classe: Coordenada e EntidadeCaixeiro 45](#_Toc468315042)

[Classe: Log de execução de GerenciadorCaixerito.run() 46](#_Toc468315043)

[EntidadeSubAlgEvo 48](#_Toc468315044)

[Execução do sugestor 52](#_Toc468315045)

[Servlet de sugestão. 54](#_Toc468315046)

SUMÁRIO

[1 INTRODUÇÃO 12](#_Toc468232649)

[1.1JUSTIFICATIVA 12](#_Toc468232650)

[1.2 OBJETIVOS 13](#_Toc468232651)

[1.2.1 Geral 13](#_Toc468232652)

[1.2.2 Específicos 13](#_Toc468232653)

[2 FUNDAMENTAÇÃO 14](#_Toc468232654)

[2.1 O QUE É INTELIGÊNCIA ARTIFICIAL 14](#_Toc468232655)

[2.2 O QUE É ALGORITMO EVOLUTIVO 15](#_Toc468232656)

[2.3 O QUE É AUTOCOMPLETE 20](#_Toc468232657)

[2.4 PROGRAMAÇÃO ORIENTADA A OBJETOS 20](#_Toc468232658)

[2.5 JAVA 21](#_Toc468232659)

[2.6 UML 23](#_Toc468232660)

[2.7 HTML 24](#_Toc468232661)

[2.8 CSS 27](#_Toc468232662)

[2.9 IDE 27](#_Toc468232663)

[2.10 BANCO DE DADOS 28](#_Toc468232664)

[2.10.1 MER 29](#_Toc468232665)

[3 DESENVOLVIMENTO 31](#_Toc468232666)

[3.1 ESCOPO 31](#_Toc468232667)

[3.2 CASO DE USO 31](#_Toc468232668)

[3.3 DIAGRAMAS DE ATIVIDADE 32](#_Toc468232669)

[3.3.1 Apresenta Sugestões 32](#_Toc468232670)

[3.3.2 Distribuição do Trecho de Texto 33](#_Toc468232671)

[3.3.3 Gerenciador de Sugestões 33](#_Toc468232672)

[3.3.4 Elimina Sugestores de Baixo Fitness 34](#_Toc468232673)

[3.3.5 O Gerenciador Reproduz as Entidades com Maior Fitness 34](#_Toc468232674)

[3.4 DIAGRAMA DE CLASSES TODO🡪MOSTRAR CÓDIGO DAS FUNCIONALIDADES DO SISTEMA 35](#_Toc468232675)

[3.5 MER 37](#_Toc468232676)

[3.6 BIBLIOTECAS 41](#_Toc468232677)

[4 IMPLEMENTAÇÃO 47](#_Toc468232678)

[5 CONCLUSÃO 49](#_Toc468232679)

[5.1 IMPLEMENTAÇÕES FUTURAS 49](#_Toc468232680)

[TODO: PEGAR DOS SLIDES 49](#_Toc468232681)

[6 REFERÊNCIAS 50](#_Toc468232682)

1 INTRODUÇÃO

Os conceitos de Inteligência artificial, software evolutivo e autocomplete não são novos e muito provável, já se cruzaram antes, possivelmente com resultados mais efetivos do que os dispostos nesse trabalho.

IA (Inteligência artificial) segundo Gongora (2007), é um conceito de comportamento inteligente em construtos humanos, buscando elevar isso à réplica de um comportamento humano.

Mas o desenvolvimento desse trabalho não tem objetivos práticos, apenas interessando-se na experimentação e pesquisa de uma alternativa, construindo um código que possa ser reutilizado. Pois o foco não está voltado para o autocomplete em si, mas sim para a metodologia utilizada na sua construção.

1.1JUSTIFICATIVA

Existem hoje no mercado inúmeras soluções para autocorrect e autocomplete, algumas já embutidas nos dispositivos ou softwares, como o autocomplete de um dispositivo Android ou o autocomplete do Google que também é um autocorrect. Estes dois termos, em suma, significam quando seu celular, tablet ou computador tenta prever o que você está escrevendo. A ideia em si é simples, perceber um padrão de escrita do usuário e correspondendo a ele sugerir correções e completar as palavras digitadas.

As soluções mais modernas para este quesito usam de inteligência artificial avançada para compartilhar os vícios linguísticos de diversos usuários, até mesmo sugerindo palavras que este ou aquele usuário nunca usaram. Isso acontece por que outros usuários já a usaram em contextos equivalentes. Por outro lado, não raro, as sugestões fazem com que as frases percam o sentido.

A proposta deste projeto é um sugestor para completar, mas não corrigir as palavras. Ele tem algumas características distintas, como por exemplo ele não faz uso de dicionários pré-estabelecidos, fazendo a construção de um dicionário interno conforme uso. Alguns dos problemas que que esta situação cria são erros ortográficos provenientes do usuário sendo replicados conforme o uso, e em um primeiro momento não haverá sugestões funcionais.

Os sugestores mais simples podem ser encontrados no Excel e no notepad++. O funcionamento destes usa como base as palavras digitadas no mesmo arquivo que se está digitando. Ao usar muitas vezes o termo “procrastinação” em um arquivo qualquer sendo editado em uma destas plataformas, ao começar a digitar “propriedade” a primeira sugestão seria “procrastinação”.

Os sugestores mais complexos como os usados em celulares fazem algo semelhante a isso, mas com uma abrangência maior (trabalhando com frases e expressões) e uma maior largura na fonte dos dados (usando de todas as frases digitadas).

1.2 OBJETIVOS

A intenção do grupo é fazer a construção de um protótipo de um aplicativo que sugira possíveis continuações para uma frase. E evoluam as sugestões com uma seleção artificial e uma reprodução espelhada à biológica.

1.2.1 Geral

Construir um protótipo de inteligência artificial que evolua na sugestão de palavras para continuidade de frases. Com um código pseudoaleatório para imitar a evolução biológica.

1.2.2 Específicos

* Inserir novas palavras conforme uso.
* Mesclar eficientemente os códigos de dois ou mais objetos, adicionando um fator aleatório.
* Simular o uso sem necessidade de uma pessoa.
* Manter em uso os objetos cujos genomas sejam mais eficientes, tirar os menos eficientes, e repopular com mesclas dos mais eficientes.

1.3 SESSÕES

Esse trabalho segue em mais seis sessões, cada uma explicando uma fração do todo:

* 1 INTRODUÇÃO
  + Explicação simplificada do trabalho, demonstrando a justificativa, e os objetivos.
* 2 FUNDAMENTÇÃO
  + Descrição das referências feitas e os conceitos dessas referências que serão utilizados a diante no trabalho.
* DESENVOLVIMENTO
  + Planejamento, diagramas e bibliotecas utilizados na implementação.
* IMPLEMENTAÇÃO
  + Implementação do projeto, construção do software.
* COMCLUSÃO
  + Conceitos adquiridos, resultados obtidos, análise das falhas, outras implementações possíveis e conclusões finais.
* REFERÊNCIAS
  + Lista dos artigos e sites utilizados para o desenvolvimento do projeto.

2 FUNDAMENTAÇÃO

Para um melhor entendimento da implementação que foi feita os conceitos abordados estão embasados e explicados nesta seção.

2.1 O QUE É INTELIGÊNCIA ARTIFICIAL

A inteligência artificial segundo Gongora (2007), é um conceito de comportamento inteligente em construtos humanos, buscando elevar isso à réplica de um comportamento humano.

“Pretende-se, com ela, capacitar o computador de um comportamento inteligente. Por comportamento inteligente devemos entender atividades que somente um ser humano seria capaz de efetuar. Dentro destas atividades podem ser citadas aquelas que envolvem tarefas de raciocínio (planejamento e estratégia) e percepção (reconhecimento de imagens, sons, etc.), entre outras. ” – (GANGORA, 2007)

Isto é, um software que apresenta comportamentos tipicamente humanos, como identificar um rosto em uma foto, são modelos de inteligência artificial.

Também segundo Gongora (2007) IA:

“É um conceito relativo à construção de estruturas cognitivas do ser humano, responsáveis pela formação da razão, característica peculiar frente aos demais animais. Como o ser humano é o único animal racional, diz-se que ele é o único ser inteligente. Há estudos que atribuem o conceito de inteligência a outros animais e vegetais. Mas, obviamente, não é um conceito comparável ao da inteligência humana. É, isso sim, um conceito relativo à análise em questão: esta inteligência irracional seria a capacidade de adaptação de um ser vivo às circunstâncias de seu meio. Desta forma, poderemos utilizar este conceito para a máquina, definindo, então, uma inteligência de máquina. ”

Portanto, o conceito de inteligência de máquina é também aplicável aos objetivos propostos para esse trabalho, como no tratamento de novas palavras, adicionando-as aos dicionários.

Outros conceitos descritos por Gongora (2007) são as tarefas especializadas, que “possui objetivos e aplicações bem específicos, dentro de um universo delimitado. ”

“Quando queremos realizar aplicações mais práticas, estamos trabalhando com tarefas especialistas, que são aquelas aplicadas em alguma profissão, resultado da síntese do conhecimento de especialistas no assunto. Daí surge o termo sistemas especialistas”. Gongora (2007)

E as tarefas formais:

“Há um conjunto de tarefas que exige análises de possibilidades para chegar-se a uma solução. Este tipo de tarefas são chamadas de formais por terem uma aplicabilidade apenas em áreas fora do mundo real, tais como jogos e problemas matemáticos. No caso de jogos, torna-se necessária a organização das ações e reações dos jogadores, todas elas previsíveis e, portanto, manipuláveis dentro de regras claras. Nos problemas matemáticos, é semelhante, havendo regras para a construção de formas geométricas, encadeamento de expressões de lógica e derivação no cálculo integral, entre outras aplicações. Estas são, portanto, aplicações situadas dentro de contextos abstratos com variáveis previsíveis. ”

Aplica-se ao problema proposto a segunda descrição, isso é as tarefas formais. É um problema com regras delimitadas, onde a interação do usuário se aplica dentro de um universo pequeno, onde tudo o que ele poderá fazer é a introdução de palavras e a seleção de sugestões.

2.2 O QUE É ALGORITMO EVOLUTIVO

Zuben (2000) descreve os conceitos da aplicação de um algoritmo (ou software) evolutivo na prática da seguinte forma:

“O software evolutivo é aplicar o processo de evolução natural como um paradigma de solução de problemas, a partir de sua implementação em computador”.

Esse paradigma baseia-se no conceito de comportamentos pseudo-aleatórios, e seleção das respostas mais próximas de resultados desejáveis. A modelagem de um algoritmo evolutivo, como descrito no artigo supracitado é baseada em uma matriz com valores aleatórios, que podem ou não respeitar regras de ordenação obrigatória, limite de valor ou até mesmo tipagem.

“A computação evolutiva deve ser entendida como um conjunto de técnicas e procedimentos genéricos e adaptáveis, a serem aplicados na solução de problemas complexos, para os quais outras técnicas conhecidas são ineficazes ou nem sequer são aplicáveis. ” (ZUBEN, 2000).

Como descrito, trata-se de um conceito útil para situações em que a programação convencional, buscando resolver o problema por meio de um computador da mesma forma que um ser humano faria manualmente, não se aplica ou falha ao ser aplicada.

Na situação de adequar-se à um comportamento tão humano quanto a sugestão de palavras, seria impraticável o uso de programação convencional, portanto a metodologia escolhida foi o algoritmo (ou software) evolutivo.

Outros conceitos descritos por Zuben (2000):

* Genes: Na Programação evolutiva, os genes serão valores de tipagem variante, que determinam um comportamento.
* Cromossomo: “estrutura nucleoprotéica formada por uma cadeia de DNA, sendo a base física dos genes nucleares, os quais estão dispostos linearmente. Cada espécie apresenta um número característico de cromossomos. ”
* *Crossover*: "(recombinação): consiste na troca (evento aleatório) de material genético entre dois cromossomos." Na prática, um software evolutivo utiliza esse conceito mesclando o genoma de dois ou mais indivíduos para formar um terceiro, que terá características de ambos, mas não será idêntico a nenhum.
* Genoma: "como muitos organismos apresentam células com mais de um cromossomo, o genoma é o conjunto de todos os cromossomos que compõem o material genético do organismo." Aplicado à metodologia evolutiva de resolução de problemas, o genoma é tido como uma matriz de cromossomos, com uma ordem e valores bem definidos, definem o comportamento final de um indivíduo.
* Genótipo: "Indivíduos e espécies podem ser vistos como uma dualidade entre seu código genético (genótipo)" Isso é, o código genético em si, não o comportamento determinado por ele.
* Fenótipo: "características comportamentais, fisiológicas e morfológicas (fenótipo)" Isso é, os comportamentos definidos pelos genes, não os genes em si.
* Pleiotropia: "um único gene pode afetar diversos traços fenotípicos simultaneamente (pleiotropia)" Um gene pode determinar dois comportamentos distintos e simultâneos.
* Poligenia: "uma única característica fenotípica pode ser determinada pela interação de vários genes (poligenia)." Vários genes, interagindo entre si, podem determinar uma característica única.
* *Fitness*: "uma função que mede a adequação relativa de cada indivíduo frente aos demais" A medida dada à adequação do comportamento tido pelo individuo frente ao problema a se resolver, que deve ter como característica uma universalidade, onde dois objetos expostos ao mesmo problema e com o mesmo fenótipo devem ter valores iguais.
* Transformação unária: “Existem transformações unárias (mutação) que criam novos indivíduos através de pequenas modificações de atributos em um indivíduo”. Na prática, é alguma alteração de um gene em um indivíduo, adicionando uma nova possibilidade ao universo de valores a se utilizar.
* Seleção de indivíduos: “O algoritmo genético clássico utiliza um esquema de seleção de indivíduos para a próxima geração chamado roulettewheel”. Essa seleção é o que define quais indivíduos permanecerão para a próxima geração, fazendo com que a evolução aconteça. É o análogo à seleção natural, quase sempre baseando-se no *fitness*.

No artigo também são exemplificados várias formar de *crossover* e mutação (Transformação unária). Tais como:

* *Crossover* de um ponto, onde a matriz genética de dois indivíduos é dividida a partir de um ponto de forma que um novo é formado com a primeira parte do primeiro genoma e a segunda parte do segundo genoma e outro com o inverso, a segunda parte do primeiro genoma e a primeira parte do segundo genoma. O ponto pode ser fixo, flutuante, ou ainda obedecer algumas regras, dependendo do problema a ser solucionado.
* *Crossover* uniforme, onde para cada gene é decidido de forma aleatória qual genitor fornecerá o valor.
* *Crossover* de dois pontos, que como o nome sugere é uma modelagem do *crossover* de um ponto onde há mais de um ponto de corte no genoma dos genitores.
* *Crossover* OX, que funciona com o mesmo principio do *crossover* de um ponto, mas com a substituição de valores repetidos pelos ausentes no genoma. Por exemplo, se houverem oito possíveis valores para os genes, mas ao executar o corte um individuo tem o valor ‘um’ repetido e o ‘dois’ ausente (nesse caso haveriam oito genes por indivíduo), o um dos valores ‘um’ seria substituído pelo valor ‘dois’.

TODO: CITAÇÃO LONGA🡪

Zuben (2000) ainda complementa: “não há nenhum operador de *crossover* que claramente apresente um desempenho superior aos demais. Uma conclusão a que se pode chegar é que cada operador de *crossover* é particularmente eficiente para uma determinada classe de problemas e extremamente ineficiente para outras. ”

Quanto aos operadores de mutação, o artigo também apresenta a descrição de algumas opções:

* “Considerando codificação binária, o operador de mutação padrão simplesmente troca o valor de um gene em um cromossomo”
* “O operador para mutação uniforme seleciona aleatoriamente um componente k ∈ {1, 2, ..., n} do cromossomo x = [x1 ... xk ... xn] e gera um indivíduo x′ = [x1 ... x′k ... xn] , onde xk′ é um número aleatório” Isso é, ao mutar um indivíduo, esse operador altera um gene aleatório para ter um valor aleatório.

Outro ponto descrito cujas abordagens são múltiplas é a seleção dos indivíduos para a geração seguinte:

* Roulettewheel: “O roulettewheel atribui a cada indivíduo de uma população uma probabilidade de passar para a próxima geração proporcional ao seu *fitness* medido, em relação à somatória do *fitness* de todos os indivíduos da população. Assim, quanto maior o *fitness* de um indivíduo, maior a probabilidade dele passar para a próxima geração.”
* Rank: “A seleção baseada em rank [...] utiliza as posições dos indivíduos quando ordenados de acordo com o *fitness* para determinar a probabilidade de seleção. Podem ser usados mapeamentos lineares ou não-lineares para determinar a probabilidade de seleção. Para um exemplo de mapeamento não-linear, veja MICHALEWICZ (1996). Uma variação deste mecanismo é simplesmente passar os N melhores indivíduos para a próxima geração.”
* Seleção por diversidade: “são selecionados os indivíduos mais diversos da população.”
* Seleção bi-classista: “são selecionados os P% melhores indivíduos e os (100 − P)% piores indivíduos.”
* Seleção aleatória: “são selecionados aleatoriamente N indivíduos da população. Podemos subdividir este mecanismo de seleção em: Salvacionista: seleciona-se o melhor indivíduo e os outros aleatoriamente. E não-salvacionista: seleciona-se aleatoriamente todos os indivíduos.”

TODO: 🡨 END, ESPECIFICAR AUTOR DA CITAÇÃO ABAIXO

“Estes mesmos mecanismos de seleção podem ser adaptados para selecionar também os indivíduos que irão sofrer *crossover* e mutação. Por exemplo, usando a seleção bi-classista, é possível selecionar os indivíduos que, ao se reproduzirem, irão gerar os indivíduos da próxima geração. ”

2.3 O QUE É AUTOCOMPLETE

Autocomplete de acordo com Computer Hope (2016), é o que faz com que sugestões plausíveis apareçam conforme o usuário digita algo, por exemplo você escreve “c” e então aparecera uma lista com “casa”, “carro”, “casaco a venda”, “vende-se carro”, ou seja ao escrever “c” será procurada uma ou mais informações que contenham a letra “c”, sempre mostrando as mais utilizadas, comuns e plausíveis.

Isso ajuda o usuário e o deixa mais confortável ao preencher um formulário. Ele é importante para deixar o preenchimento desses campos mais dinâmico, mais rápido, afinal todos os usuários sempre estão em busca de um meio para fazer suas pesquisas de forma mais rápida, para poupar tempo tal que será usado de forma mais produtiva em outro momento.

2.4 PROGRAMAÇÃO ORIENTADA A OBJETOS

A programação Orientada a Objetos de acordo com Nery (2016) é um paradigma, modelo, padrão de programação de computadores, que usa de conceitos voltados à Objetos e Classes como elementos centrais para representar e processar os dados usados nos softwares. A ideia da POO é que poderíamos construir um programa usando conceitos e abstrações do mundo real, como objetos.

O princípio da abstração é a capacidade de abstrair a complexidade de um sistema e se concentrar apenas em partes do mesmo, por exemplo ” um médico torna-se um especialista em algum órgão do corpo, digamos que seja o coração. Ele abstrai sem desconsiderar as influências dos outros órgãos e foca apenas sua atenção nesse órgão”. (NERY, 2016)

Nery (2016) afirma que os objetos são usados para representar entidades do mundo real, se olharmos com atenção ao redor do nosso espaço, será possível ver várias entidades que podem ser representadas como objetos no nosso programa, as escolas e seus alunos podem ser vistas como objetos, os objetos têm dois pontos principais, as características ou seja, atributos pelos quais os identificamos e as finalidades para as quais os utilizamos um objeto pode ter comportamentos associados, chamados de métodos, por exemplo, uma pessoa pode andar, correr ou dirigir.

Nery (2016) explica que utilizamos de Classes para criar nossos objetos, cada classe funciona como um molde, uma forma para a criação de um dado objeto, portanto os objetos são vistos como representações completas, ou seja, instâncias das classes.

A Programação Orientada a Objetos é sedimentada sobre três pilares que provêm do princípio da abstração, sendo eles:

1. Encapsulamento: “deriva-se de cápsula, que nos lembra qualquer forma pequena que protege algo em seu interior como um medicamento. Ele ajuda a desenvolver programas com maior qualidade e flexibilidade para mudanças futuras” (NERY, 2016). O encapsulamento também é capaz de ocultar partes (dados e detalhes) de implementação interna de classes do mundo exterior.
2. Herança: é o mecanismo que permite a uma classe herdar todos os atributos e métodos de outra classe. “Quanto mais alta a classe na hierarquia, mais ela tende a ser abstrata” (NERY, 2016), ou menos definida, com menos atributos e métodos. Suponha que há uma classe Transporte, ela tem o atributo capacidade, ela pode ter uma classe filha chamada Terrestre ou Aquático que tem seus próprios atributos e métodos mais específicos
3. Polimorfismo: Nery (2016) descreve polimorfismo como a habilidade de objetos e classes responderem a mesma requisição de muitas maneiras. Por exemplo, o dono de uma fábrica de brinquedos instruiu seus engenheiros a criar um mesmo controle remoto para todos os brinquedos de sua fábrica, a única restrição era que cada brinquedo atendesse aos comandos específicos definidos pelo controle.

2.5 JAVA

De acordo com Peter (1999) em Introdução ao Java, a linguagem de programação Java em conjunto de sua plataforma, constituem um fascinante objeto de estudo, com um conjunto rico de bibliotecas para facilitar o desenvolvimento e utiliza como paradigma central a Orientação a Objetos. O Java é o resultado de um trabalho consistente de pesquisa e desenvolvimento de mais do que uma simples linguagem de programação, mas de todo um ambiente de desenvolvimento e execução de programas que exibe as finalidades proporcionadas pela orientação à objetos, pela extrema portabilidade do código produzido, pelas características de segurança que esta plataforma oferece e finalmente pela facilidade de sua integração aos outros ambientes, destacando-se a Web.

Algumas de suas características mais importantes de acordo com Peter (1999) que, agrupadas, diferenciam-na de outras linguagens de programação são:

* Orientado à Objetos: com exceção dos tipos primitivos (*int*, *float*, etc... ) tudo em Java são classes ou objetos. Java pode ser considerada orientada a objetos pois atende aos três pilares da orientação a objetos (abstração, encapsulamento e hereditariedade).
* Independente de Plataforma: considerando que o código de Java é compilado e se transforma em *bytecodes*, que são praticamente uma linguagem de máquina que é direcionada somente à JVM (*Java Virtual Machine*, máquina virtual Java).
* Sem Ponteiros: Java possui um *garbage collector* isto é, Java não permite a manipulação direta de endereços de memória nem exige que os objetos criados sejam destruídos livrando os programadores de uma tarefa complexa.
* Performance: “Como Java é uma linguagem interpretada a performance é razoável, não podendo ser comparada a velocidade de execução de código nativo. ” (PETER, 1999).
* Segurança: “Considerando, a possibilidade de aplicações obtidas através de uma rede, a linguagem Java possui mecanismos de segurança que podem, no caso de applets, evitar qualquer operação no sistema de arquivos da máquina-alvo, minimizando problemas de segurança.Isso é flexível o suficiente para determinar se a aplicação é segura especificando nessa situação diferentes níveis de acesso ao sistema-alvo”. (PETER, 1999)
* Multithreading: Java oferece meios para o desenvolvimento de softwares capazes de executar múltiplas rotinas concorrentemente, também dispõe de elementos para a sincronização das mesmas. Cada um destes fluxos de execução é o que se denomina thread, um importante recurso de aplicações mais sofisticadas.

Java é uma linguagem muito completa, oferece tipos com as especificações IEEE, suporte para caracteres UNICODE, é extensível dinamicamente, além de ser voltada por natureza ao desenvolvimento de aplicações em rede ou distribuídas.

2.6 UML

Segundo Gudwin (2010), a linguagem UML (*Unified Modeling Language*) é uma linguagem de modelagem criada visando-se a criação de modelos abstratos de processos sendo modelados. Tanto podem ser processos do mundo real, como processos de desenvolvimento de software ou ainda detalhes internos do próprio software. Assim tanto podemos utilizá-lo para descrever o mundo real, como a organização de uma empresa, como os detalhes internos que descrevem um sistema de software. A descrição de um processo envolve a determinação de duas classes básicas de termos:

* Os elementos estruturais que compõem o processo;
* O comportamento que esses elementos desenvolvem quando interagindo.

Descrito por Gudwin (2010) o UML pode ser estendido com novos tipos de diagramas criados por mecanismos chados *Profiles*. Diferentes *Profiles* podem ser construídos utilizando-se estereótipos, os *tagged values* e as restrições.

“A linguagem UML, por meio de seus diagramas, permite a definição e design de *threads* e processos, que permitem o desenvolvimento de sistemas distribuídos ou de programação concorrente. “ (Gudwin, 2010). Da mesma maneira, permite a utilização dos chamados patterns (patterns são, a grosso modo, soluções de programação que são reutilizadas devido ao seu bom desempenho) e a descrição de colaborações (esquemas de interação entre objetos que resultam em um comportamento do sistema).

“Um dos tipos de diagramas particularmente úteis para modelarmos processos são os chamados diagramas de atividades. Por meio deles, é possível especificarmos uma sequência de procedimentos que compõem um processo no mundo real. “ (GUDWIN, 2010). Diagramas de atividade podem, portanto, ser utilizados para descrever o processo de desenvolvimento de software (por exemplo, o processo Unificado).

Uma outra característica do UML é a possibilidade de fazermos uma descrição da hierarquia dos processos. Com isso, podemos fazer melhorias no nosso modelo, descrevendo o relacionamento entre as entidades em diferentes níveis de abstração.

Para fazer essa melhoria, temos de abstrair os componentes de um modelo pelas suas interfaces. Desta forma, os componentes são criados somente em função de suas entradas e saídas. Essa característica permite ao UML um desenvolvimento incremental do modelo.

2.7 HTML

HTML (Hiper Text Markup Language) descrito por Eis (2011), é uma linguagem de marcação usada como base para sites web, com ela nós marcamos elementos para mostrar quais informações a página exibe, exemplo do uso da tag<h1>:



1 Exemplo de Tag de HTML

Como se pode ver há um texto entre duas marcações, as marcações <h1> não vão aparecer na página mas vão modificar o texto que há entre elas.

Para cada marcação de abertura sempre há uma de fechamento, no caso para <p> será </p>. Também há outras marcações como <h1> para títulos.

Uma das principais características do HTML se reflete no fato de se um programa navegador não "entender" um determinado comando, este é ignorado e não *é* apresentado, não originando mensagem de erro e afetando minimamente o restante do documento.

Assim, resumidamente, quando se digita um endereço de um site, o navegador:

* Contata o servidor de DNS e descobre onde está o computador que hospeda o site desejado;
* Envia o pedido de cópia do(s) arquivo(s) que está naquele endereço;
* Então, o servidor web analisa e trata o pedido e responde ao navegador com um arquivo de texto;
* O navegador obedece o texto e constrói a página na tela do cliente;
* A pessoa *vê*, em seu monitor, a página web solicitada;

“[...] nem sempre diferentes navegadores exibem a mesma apresentação em cada página. Ou seja, os detalhes codificados no HTML podem ser suficientes para um deles mas não suficientes para outro” (Eis, 2011).

Assim, cada navegador poderá interpretar os dados de uma forma um pouco diferente. Portanto, quanto mais perfeitamente descrita a página for, maiores serão as chances do documento ser interpretado da mesma forma por diversos navegadores.

O HTML não foi criado para controlar a aparência das páginas, ele apenas indica ao navegador o que é o conteúdo da página, quais arquivos ela contém e onde estão usando as marcações.

Porém o HTML é estático, sem movimento ou qualquer outra coisa qe atraia a atenção do usuário, mas ele pode ser dinamizado por linguagens como JavaScript e CSS.

É importante lembrar que o HTML é um texto que pode apresentar sons, vídeos, imagens e outras aplicações. Ou seja, os dados que podem ser transferidos podem ser de qualquer tipo.

Porém, o que realmente caracteriza o hipertexto é que pode se criar links, isto é, ligações para outros arquivos (textos, imagens, sons, vídeos). Assim, os links possibilitam a "navegação", tanto dentro de um arquivo, site, de um ponto para outro, ou entre arquivos diferentes, que podem estar em computadores também diferentes, e que podem estar localizados proximamente ou estar extremamente distantes no nosso plano terrestre.

## 2.8 CSS

De acordo com Guimarães e Santos (2008), Cascading Style Sheets (CSS) é usado para garantir uma formatação homogenia e uniforme nas páginas de um website, ou seja, CSS é um padrão de formatação para páginas, o que nos permite sair das limitações de layout e legibilidade de um arquivo HTML.

O CSS possui uma regra simples que pode ser dividida em duas partes, seletor e declaração. O seletor, como o próprio nome já o denuncia, é o que liga elemento à declaração. A declaração possui duas subpartes, propriedade e valor. Propriedade é uma qualidade ou uma característica que um elemento deve possuir.

Guimarães e Santos (2008) também descrevem mais a fundo os três mantras que devem ser seguidos ao se aplicar o CSS nas páginas Web, que são respectivamente:

1. Inline: em linha, diretamente no elemento que você deseja afetar, você pode chamar uma propriedade *style* e escrever CSS que será atribuído à propriedade;
2. Embedding: interno, incorporado, embutido, nesse caso você usa uma marcação de texto do HTML( <style> ) e dentro dela você pode escrever livremente seus seletores com suas declarações, que afetarão somente o arquivo em que estão sendo escritas;
3. Linking: externo, esse é o mantra usado por programadores experientes, pois permite a reutilização dos estilos para múltiplas páginas sem poluir visualmente seu texto HTML. Essa referência ao arquivo CSS é feita a partir de uma marcação de texto do html( <link> ), nele você insere o caminho a partir do arquivo em que você está desenvolvendo está situado no servidor, computador, dentre outros.

2.9 IDE

IDE, ou Ambiente Integral de Desenvolvimento em tradução livre descrito por Novaes (2014), é um software criado com a finalidade de facilitar a vida dos programadores. Neste tipo de aplicação estão todas as funções necessárias para o desenvolvimento desde programas de computador a aplicativos mobile, assim como alguns recursos que diminuem a ocorrência de erros nas linhas de código.

Novaes (2014) compara a ideia de que antigamente se aprendia matemática na escola, mas raramente fazia-se manualmente e acabava-se por usar uma calculadora, imaginando o cálculo como um software e as IDEs como calculadoras é possível notar que ambas possuem o mesmo objetivo de certa forma, poupar tempo gasto.

Novaes ainda fala sobre uma das principais vantagens dos IDEs que é a possiblidade de compilar bibliotecas completas de linguagem. Outra função bastante comum neste tipo de programa são os *debuggers*, que apontam os erros que ocasionalmente podem ocorrer ao escrever o código, principalmente os de sintaxe. Algumas IDEs também possuem o autocomplete para classes, métodos, propriedades, dentre outros.

2.10 BANCO DE DADOS

Segundo Date (2004), “Um banco de dados é uma coleção de dados persistentes, usada pelos sistemas de aplicação de uma determinada empresa”. Em outras palavras, um banco de dados é um local onde são armazenados dados necessários à manutenção das atividades de determinada organização, sendo este repositório a fonte de dados para as aplicações atuais e as que vierem a existir.

Para Elmasri e Navathe (2011), na expressão Banco de Dados estão subentendidas as propriedades abaixo:

“Um banco de dados representa algum aspecto do mundo real, às vezes chamado de minimundo ou de universo de discurso (UoD – *Universe of Discourse*). As mudanças no minimundo são refletidas no Banco de Dados. Um banco de dados e uma coleção logicamente coerente de dados com algum significado inerente. Uma variedade aleatória de dados não pode ser corretamente chamada de banco de dados. Um banco de dados é projetado, construído é populado com dados para uma finalidade específica. Ele possui um grupo definido de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados. ”

Portanto um banco de dados nada mais é do que um conjunto de dados relacionados, criado com dado objetivo que atende uma comunidade de usuários.

2.10.1 MER

De acordo com O. K. (p. 22), MER é um modelo de dados conceitual de alto nível, ou seja, seus conceitos foram feitos de forma que praticamente qualquer pessoa possa compreender o que se passa em um. Atualmente sendo usado durante a projeção do banco de dados.

Descrito por O. K. (p. 22) O MER dispõe de alguns conceitos básicos para sua compreensão:

* Entidades: são representações de coisas do mundo real independentes, podem ser pessoas, empresas, escolas, etc… Cada Entidade possui características particulares, chamadas de Atributos.
* Atributos: são as características que descrevem uma entidade, por exemplo, a entidade aluno tem nome, rg, cpf, idade, ano, etc…
* Relacionamento: é uma estrutura que indica uma associação entre instâncias de duas ou mais entidades.
* Relacionamento Binário: é a relação de dois conjuntos de entidades distintos.
* Relacionamento Ternário: é a relação de três conjuntos de entidades distintos.
* Cardinalidade: Indica o número de instâncias de entidades que podem estar associadas umas às outras através de um relacionamento, por exemplo, um aluno pode estar em várias escolas, e uma escola pode ter vários alunos, ou seja, é uma cardinalidade de N para N.
* Cardinalidade 1:1 : Uma entidade no grupo A está associada no máximo a uma entidade no grupo B e uma entidade em B está associada no máximo a uma entidade em A, por exemplo, uma cadeira tem espaço pra 1 pessoa, enquanto 1 pessoa só pode se sentar em uma cadeira.
* Cardinalidade 1:n : Uma entidade em A está associada a qualquer número de entidades em B, enquanto uma entidade em B está associada no máximo a uma entidade em A, por exemplo um cartão de crédito pode ter apenas 1 dono enquanto uma pessoa pode ter N cartões de crédito.
* Cardinalidade n:n : uma entidade em A está associada a qualquer número de entidades em B e vice-versa.

Portanto com o MER podemos montar nossa base de dados baseando-se em uma regra de negócios, definindo as entidades e as relações entre elas como se fossem relações na vida real, como por exemplo uma empresa que possui muitos funcionários e que cada funcionário por sua vez tenha uma função. A partir do MER você tem uma certa ideia de como o sistema vai funcionar, inclusive a ordem de cadastro e a hierarquia das entidades como usuários com perfil de administradores ou usuários com perfil de cliente.

3 DESENVOLVIMENTO

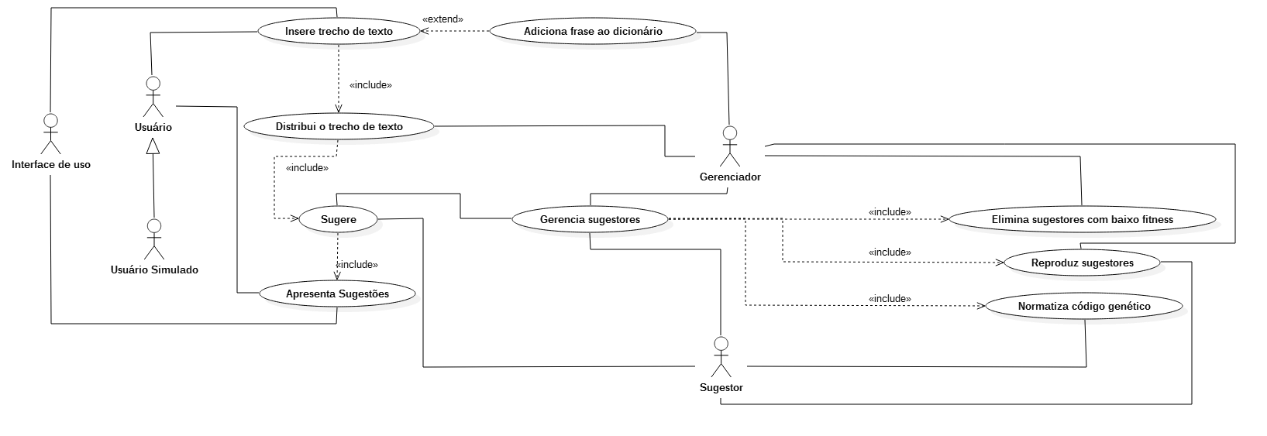
Nesse capítulo exploraremos o planejamento dado à implementação, escopo, diagramas, bibliotecas e análises do problema proposto.

3.1 ESCOPO

O desenvolvimento de um software inteligente, ainda mais no âmbito de um algoritmo evolutivo depende, inicialmente, na delimitação de um escopo. O intuito do protótipo desenvolvido é explorar as possibilidades de uso do software evolutivo em um auto-complete, sem a obrigação de ser bem-sucedido nesse processo.

A primeira definição de escopo feita foi um diagrama de casos de uso, que considera alguns atores, tais como o usuário, o usuário simulado, as diferentes entidades com soluções propostas ao problema, e gerenciadores, da interface apresentada e das próprias entidades que sugerem.

3.2 CASO DE USO



2 Diagrama de Caso de Uso

É da responsabilidade do usuário (e do usuário simulado) inserir texto base, e escolher uma ou nenhuma das sugestões apresentadas. Essas ações se dão através da interface de uso, que gerencia esses processos e nada além.

O gerenciador, um objeto que contém e controla todas as entidades do software evolutivo, doravante chamadas de sugestores, é responsável por:

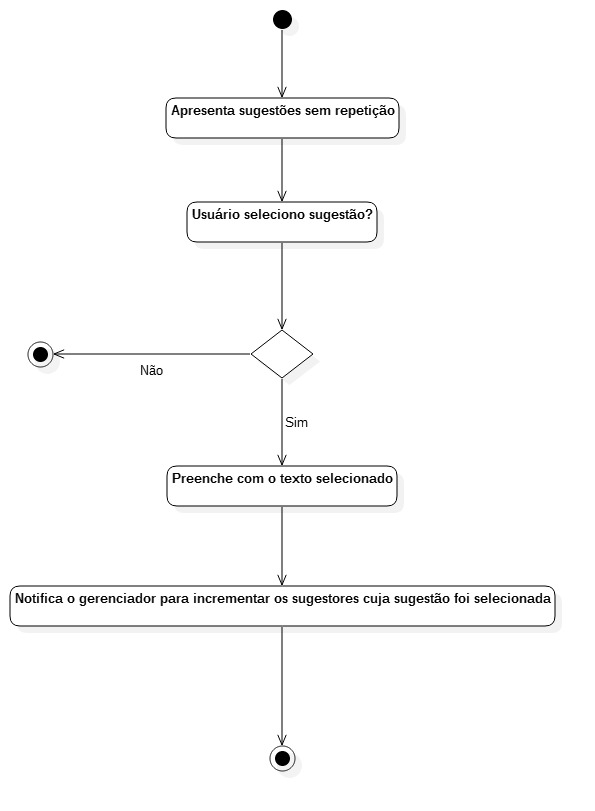
* Inserir os textos no banco de dados, construindo uma biblioteca de referências para os sugestores;
* Gerenciar (eliminar, reproduzir, executar, normalizar o código e mutar) os sugestores.

Os sugestores por si só, são responsáveis por gerar uma alternativa possível de continuidade no trecho de texto digitado pelo usuário.

3.3 DIAGRAMAS DE ATIVIDADE

Todas as ações dos atores receberam diagramas de atividades, as mais importantes seguem:

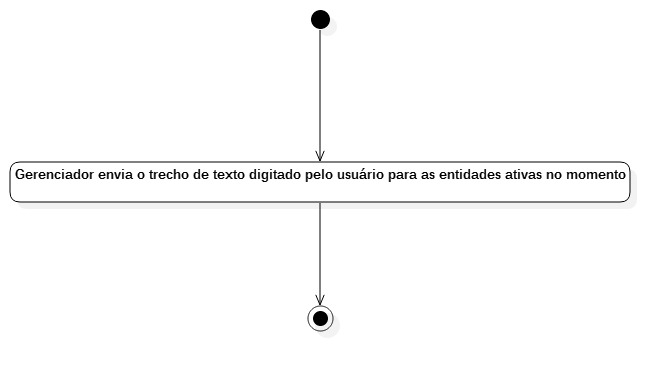
3.3.1 Apresenta Sugestões

A response (reposta) da requisição feita quando o usuário começou a digitar é uma lista com determinada quantidade de palavras, que o usuário pode escolher ou não. Caso o usuário escolha uma da lista, o texto é complementado com a opção selecionada e é enviada uma requisição para que o gerenciador incremente o sugestor cuja a sugestão foi selecionada.

3 Diagrama de Atividade - Apresenta Sugestões

3.3.2 Distribuição do Trecho de Texto

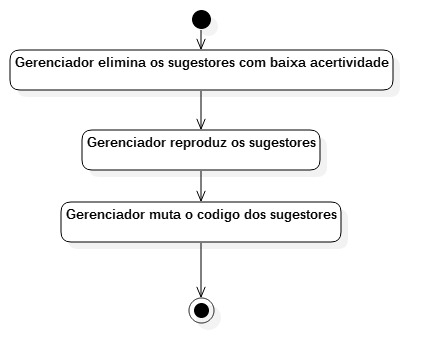
Assim que o usuário começa a digitar, todos os caracteres são enviados continuamente de maneira assíncrona para as entidades que não foram eliminadas e estão ativas.



4 Diagrama de Atividade - Gerenciador Distribui texto

3.3.3 Gerenciador de Sugestões

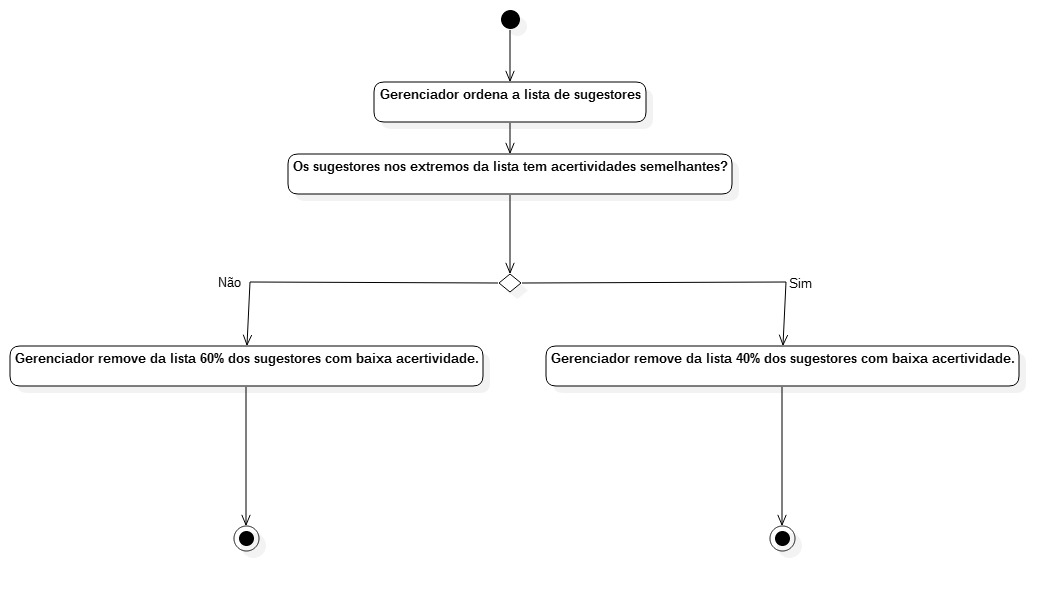
Os sugestores que dificilmente são utilizados pelo usuário são considerados sugestores de baixa atividade, o gerenciador elimina todos os sugestores com baixa atividade, reproduz o restante e efetua uma mutação no código genético de cada um deles.



5 Diagrama de Atividade - Gerenciador

3.3.4 Elimina Sugestores de Baixo Fitness

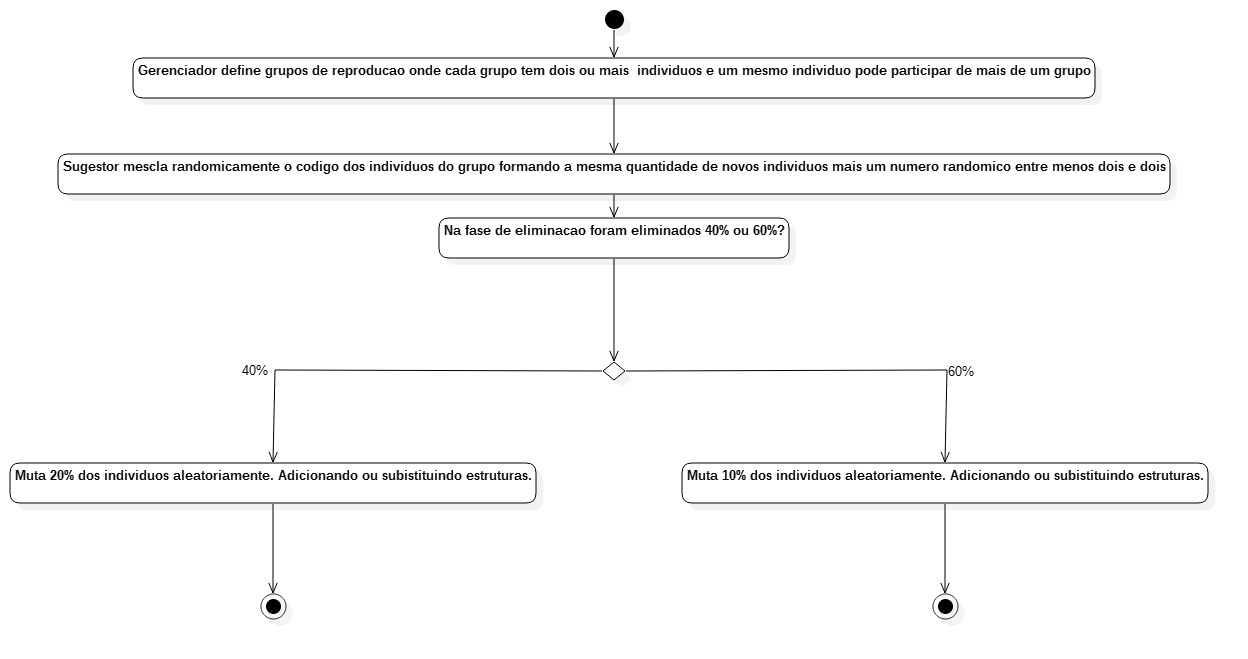
O gerenciador pega a lista utilizada pelo usuário e a ordena, logo após verifica se o primeiro e o último da lista são frequentemente utilizados, se são muito utilizados ele remove quarenta por cento da lista, priorizando os com menor *fitness* (assertividade, quantidade de usos), se não são muito utilizados ele faz o mesmo procedimento, porém a quantidade de sugestores removidos aumenta para sessenta por cento.



6 Diagrama de Atividade - Gerenciador Elimina Sugestores.

3.3.5 O Gerenciador Reproduz as Entidades com Maior Fitness

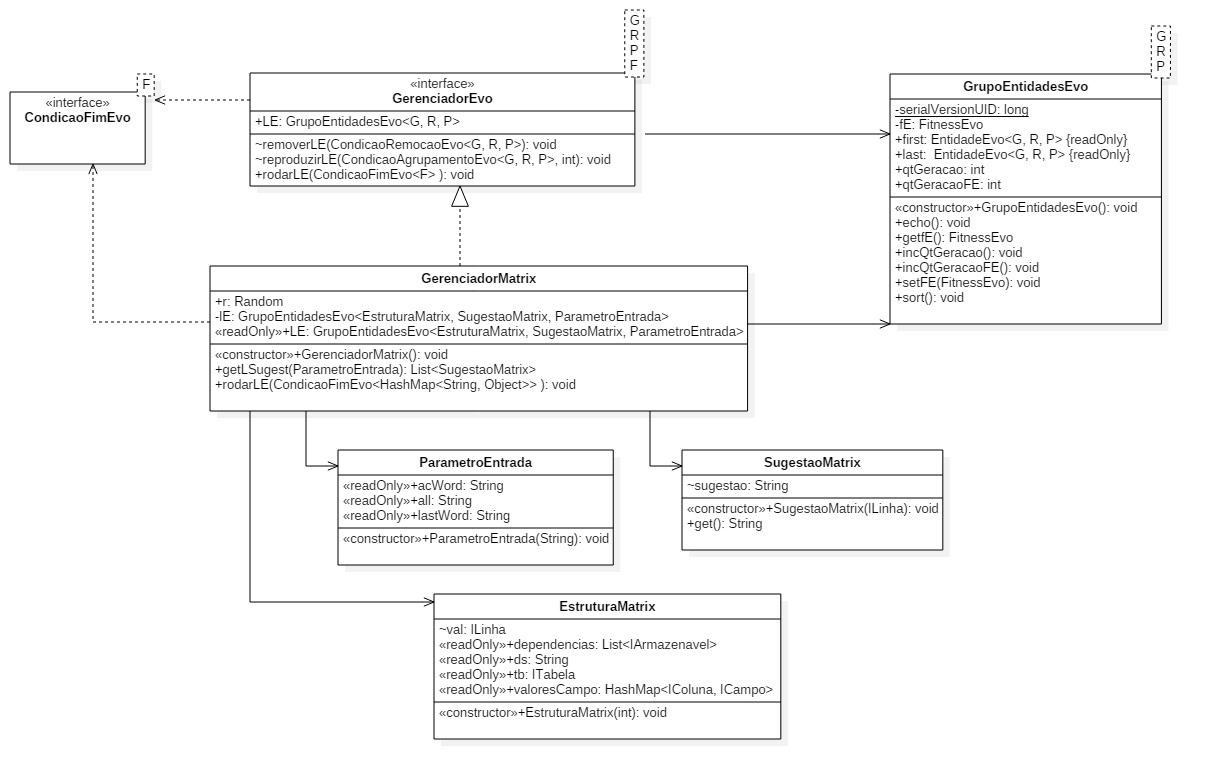
O gerenciador define grupos de reprodução, podendo cada grupo ter um ou mais indivíduos, e cada um destes indivíduos podem participar de mais de um grupo de reprodução. O sugestor mistura aleatoriamente o código dos indivíduos de cada grupo, formando a mesma quantidade de indivíduos, mais um número aleatório entre menos dois e dois. Logo após o processo de reprodução o gerenciador checa se na fase de eliminação, a quantidade dos indivíduos eliminados foi de quarenta por cento ou sessenta por cento para equilibrar o número de indivíduos. Se foi quarenta, ele muta vinte por cento dos indivíduos aleatoriamente, adicionando e removendo estruturas, senão muta somente dez por cento.



7 Diagrama de Atividade - Gerenciador reproduz entidades.

3.4 DIAGRAMA DE CLASSES

Foi definido um diagrama de classes com as entidades principais do sistema como mostrado abaixo:



8 GerenciadorMatrix

No diagrama de classes há algumas classes que precisam de uma atenção especial sendo elas:

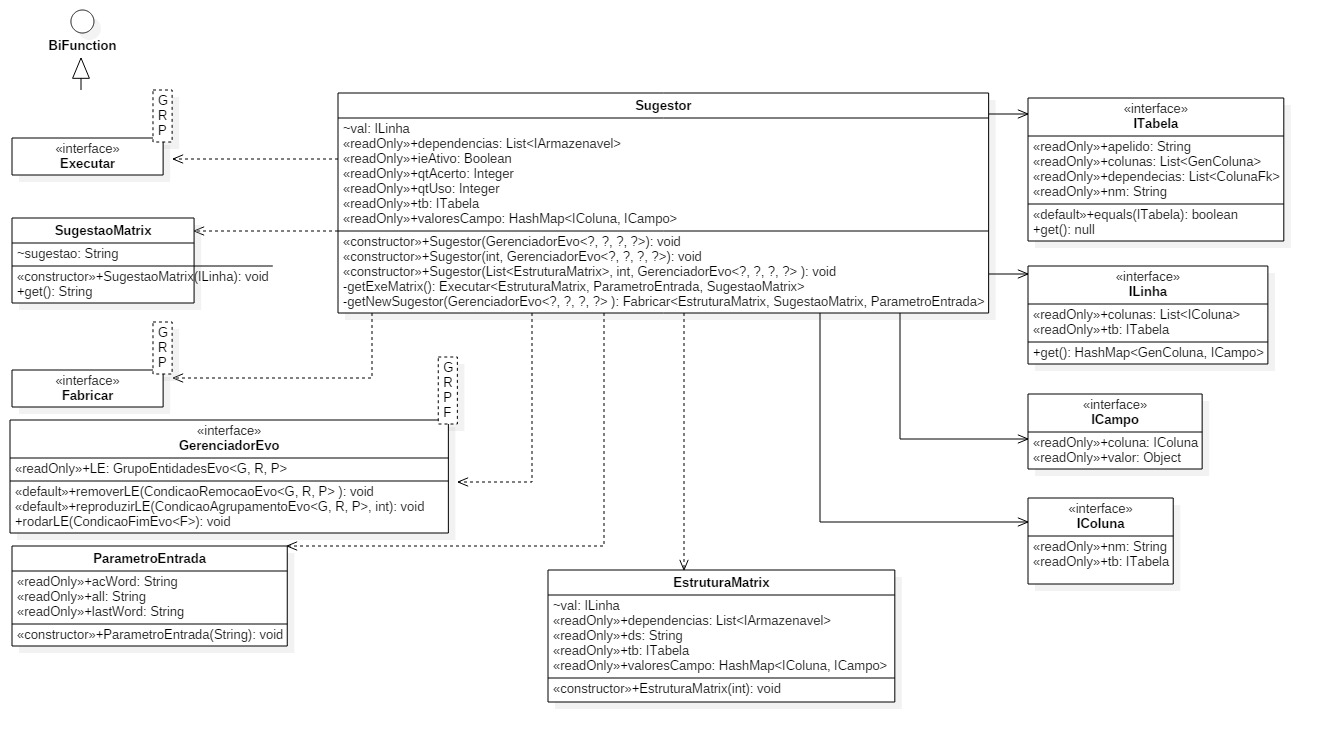
* **GerenciadorMatrix:** essa classe implementa uma interface genérica que é a GerenciadorEvo, também utilizada pelo GerenciadorCaixeiro, é uma interface que possui métodos para manipular as listas de entidades, removendo, reproduzindo e mutando elas.

O gerenciador possui um GrupoEntidadesEvo, que nada mais é que um conjunto de entidades com determinado *fitness* e número de sua geração. Esse grupo será alterado dependendo do quão assertivo ele é.

O GrupoEntidadesEvo do gerenciador possui SugestãoMatrix, que nada mais é que uma palavra que vai sair, assim como o ParametroEntrada, que vai vir à classe.

A CondiçãoFimEvo define até quando o Gerenciador vai mutar as estruturas e a lista de entidades.

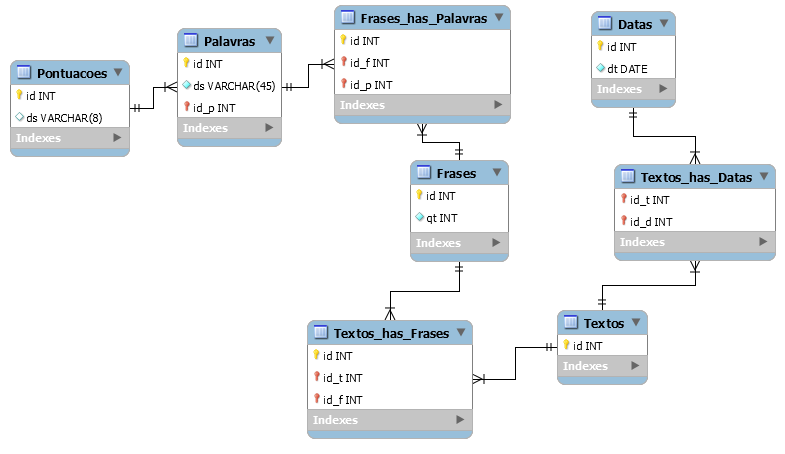
* **Sugestor:** ele utiliza as interfaces ITabela, ILInha, ICampo e IColuna, para pegar as informações de determinada tabela. A interface Fabricar é usada para criar novas EstruturasMatrix, sem o GerenciadorEvo o Sugestor não funciona, pois ele é dependente do mesmo, pois o GerenciadorEvo vai fazer maior parte das alterações nas listas de entidades, para ele filtrar e “sugerir” SugestaoMatrix.



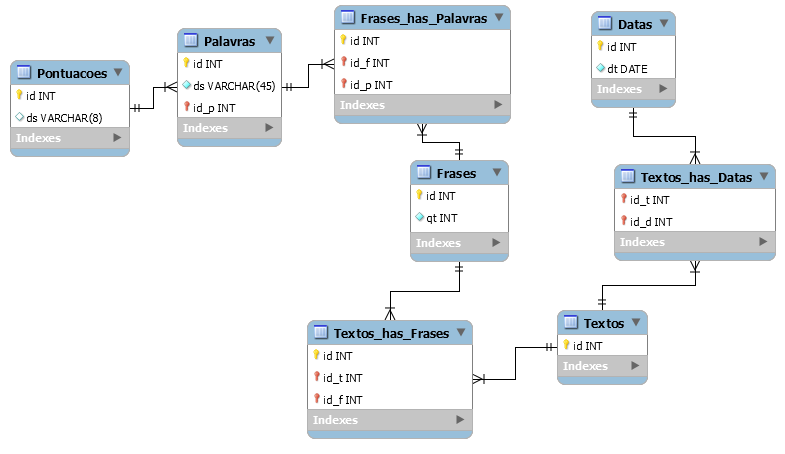
9 Sugestor

3.5 MER

Também foi definido, em termos genéricos, a modelo entidade relacionamento do banco de dados. Com divisão de palavras, frases, textos, sugestores e estruturas.



10 MER do Banco de Dados (parte 1)



11 MER do Banco de Dados (parte 2)

O intuito do uso desse modelo de relacionamentos é o melhor aproveitamento do espaço no banco de dados e a construção de diversas relações entre os diferentes componentes de um texto. A pontuação, as palavras, as frases, e os textos em si. Essa análise também visa promover a busca com uso das relações entre os componentes do texto.

Ela também se preocupa em armazenar os sugestores de forma eficaz, guardando informações sobre os genitores e as diferentes estruturas que compõe o código genético de cada sugestor. Essas, por si só, definem o sugestor.

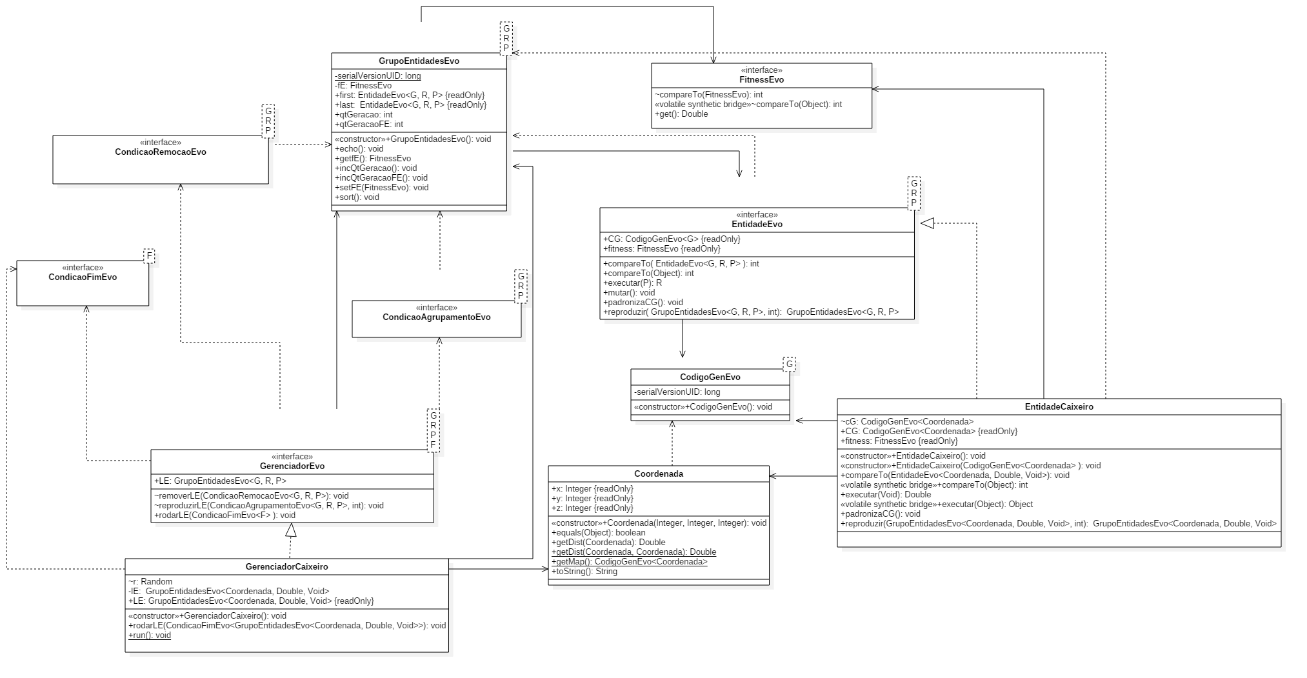
Mas essas definições não são, no entanto, o bastante para o desenvolvimento de um projeto e de um protótipo. Foi desenvolvida também a tabela de riscos, aqui representada em tópicos:

* Não conseguir forçar a evolução, pela na natureza da mesma.
  + Categoria: Tecnologia
  + Descrição: Estagnação de forma que os objetos no ambiente não retornem resultados positivos.
  + Impacto: Baixo.
  + Probabilidade: Baixa.
  + Causa: Modelagem ruim do código genético.
  + Tratativa: Alterar manualmente o código genético.
* A aplicação pode não evoluir para um lado desejável.
  + Categoria: Tecnologia
  + Descrição: Sugestões inadequadas ou impróprias.
  + Impacto: Baixo.
  + Probabilidade: Média.
  + Causa: Limitação nos ciclos de uso simulado.
  + Tratativa: Executar uso simulado com inputs diferentes dos iniciais.

Apesar de a tabela de riscos ter mais itens, foram litados aqui apenas os que realmente ocorreram, pois esses tiveram significativa importância no decorrer do projeto. A tratativa dos mesmos e suas respectivas ocorrências estarão descritas mais à frente, nos trechos em que ocorreram.

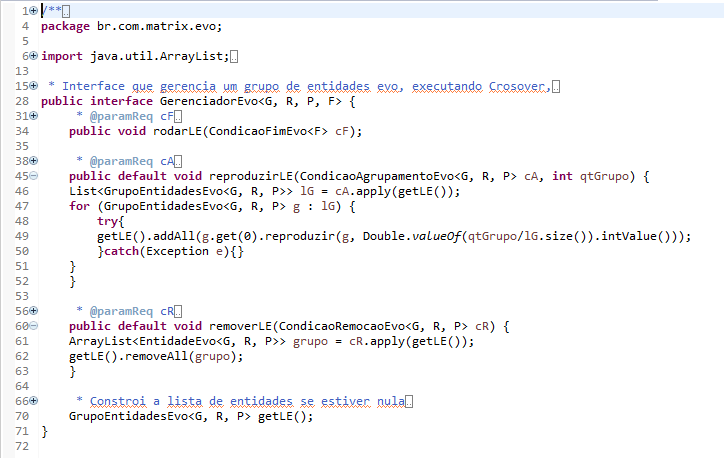
3.6 BIBLIOTECAS

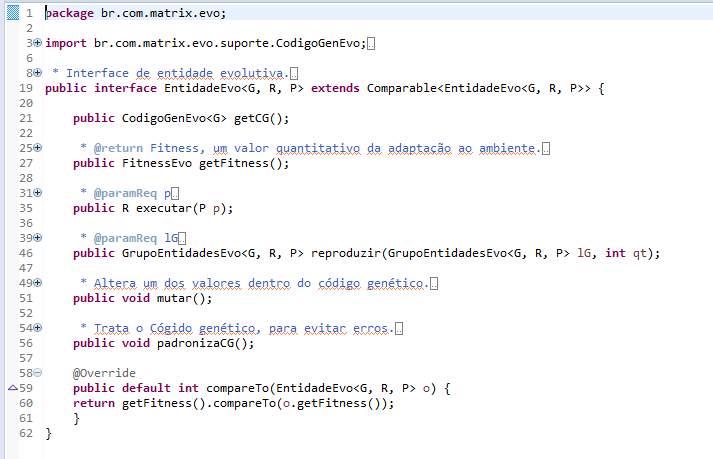
Foram construídas, na linguagem Java, uma série de classes com a função de tratar no escopo do software evolutivo, todos os problemas associados. Sem, não obstante, amarrar-se a soluções delimitadas, isso é, permitindo seu re-uso.



12 Diagrama de Classes - Caixeiro Viajante

Essa biblioteca organiza-se com base em dois componentes, o GerenciadorEvo e a EntidadeEvo. O primeiro define alguns métodos, como o que executa cada uma das entidades contidas, o que as reproduz, agrupando-as conforme uma regra recebida, e o que remove as entidades com base numa regra. O componente EntidadeEvo tem alguns métodos, responsável por mutar, normalizar e mesclar o código genético e utilizá-lo em uma execução.

13 Interface GerenciadorEvo



14 Interface EntidadeEvo

Esses tratamentos são tão genéricos quanto possível. Como por exemplo o ciclo de reprodução dessas entidades. O gerenciador recebe uma regra para fazer o agrupamento, e conforme essa regra, ele envia à primeira entidade as listas. A entidade por sua vez não define como essas listas gerarão novos indivíduos, mas possui um método abstrato, isso é, essa definição é feita em uma classe filha.

Essa biblioteca segue os padrões definidos por Zuben (2000), com alguns acréscimos:

* A tipagem do código genético, descrita por Zuben (2000) como sendo sempre booleana ou de ponto flutuante perde essa limitação, isso é, podendo receber qualquer coisa, desde um inteiro à uma classe criada pelo programador.
* Tal como a tipagem, a quantidade de alelos para o código genético de cada entidade não recebe uma limitação interna à biblioteca, sendo diferençável para cada problema.
* Na execução das entidades, elas podem receber parâmetros, fazendo que um mesmo código genético possa ter comportamentos diferentes em diferentes contextos, o que é bastante importante para o problema proposto.
* A própria normalização do código genético, com o intuito de evitar erros, recebe aqui uma importância maior do que a descrita por Zuben (2000).

Essa biblioteca também é dotada de outra característica a ser descrita, ela foi feita tão orientada a objetos quanto possível. Evitai-se ao máximo linhas de código repetidas e, por diversas vezes, criando interfaces que não propriamente assinavam métodos.

Por exemplo, ao invés de utilizar a interface do Java Function<List<EntidadeEvo<>>, List<List<EntidadeEvo<>>>> nós criamos uma interface AgrupaEntidades que a estende. Nesse caso, a Function representa a regra de agrupamento de genitores, agrupando quais entidades terão o código genético mesclado para geração de novas.

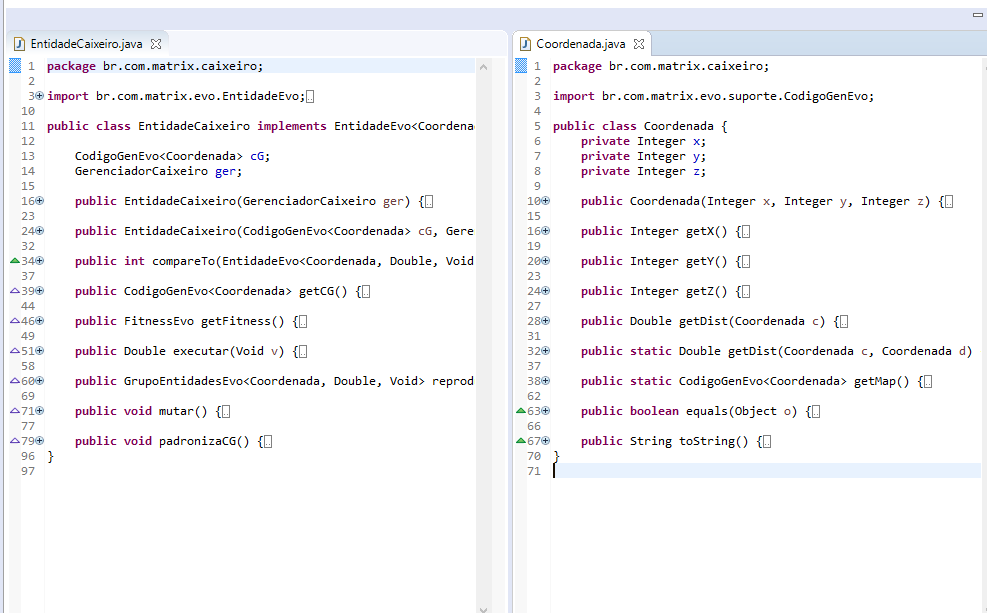
Essa parcela do software, pode-se chamar de biblioteca, foi desenvolvida em conjunto com uma aplicação no intuito de resolver o problema do caixeiro viajante. No intuito de encontrar as falhas que a análise inicial acarretaria.

O problema do caixeiro viajante é bastante antigo, e cabe aqui uma breve descrição: Um caixeiro viajante quer passar por uma dada quantidade de cidades, ele consegue ir de qualquer uma para qualquer outra, sem limitações geográficas, mas ele quer fazê-lo de forma que passe uma vez por cada cidade, retornando à primeira no final, e de forma que, a soma das distâncias percorridas seja a menor possível.

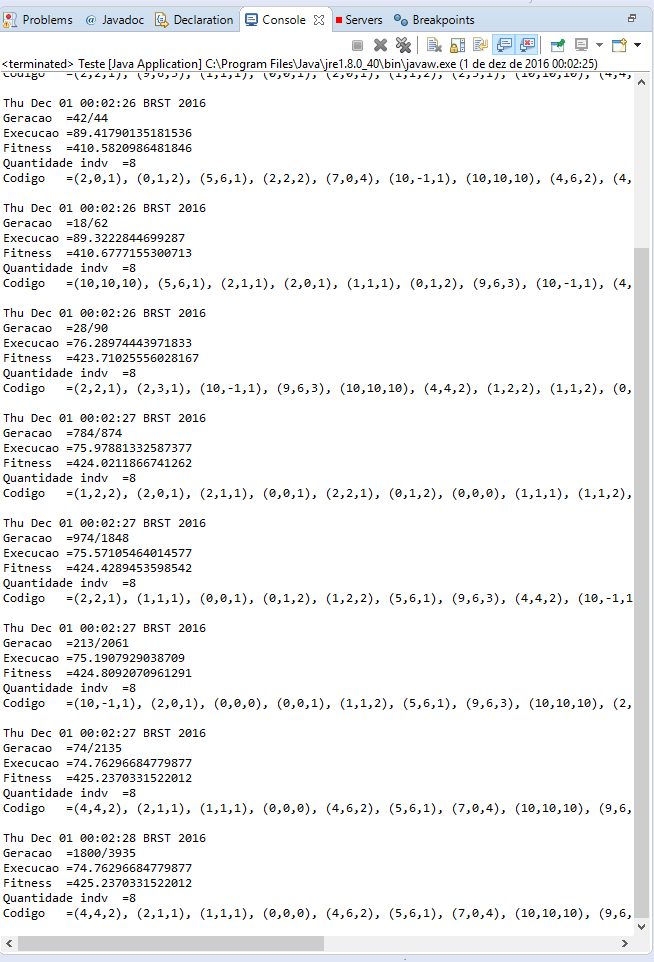
É um problema de resolução aparentemente simples, considerando seis ou sete cidades, em um plano bidimensional, mas conforme aumentamos a quantia de cidades, a resolução fica simplesmente impossível. Imagine cem cidades, a quantia total de possibilidades de rotas é noventa e nove em fatorial. Ou imagine se fosse em um plano com três dimensões. Torna-se simplesmente impossível para um ser-humano achar a solução mais otimizada.

A aplicação da biblioteca na resolução desse problema deu resultados bastante otimizados, em mais da metade das tentativas retornando uma solução aceitável, e em mais de vinte por cento dos casos sendo a solução mais otimizada que pudemos encontrar.

Foram utilizadas, conforme no exemplo descrito por Zuben (2000), as coordenadas como tipagem do código genético. A normalização serviu, nesse caso, para que ele não repetisse nenhuma cidade. A mutação foi feita trocando cidades de posição no código genético, ou blocos de cidades. A reprodução era feita dando preferência aos com maior *fitness*. O primeiro teria um contra dois de passar seu código genético, se não for ele, o próximo terá a mesma chance, depois o terceiro, e se o último falhar, volta-se ao primeiro. Obviamente, o *fitness* que é a unidade de medida de adaptação é medido pela menor soma das distâncias.

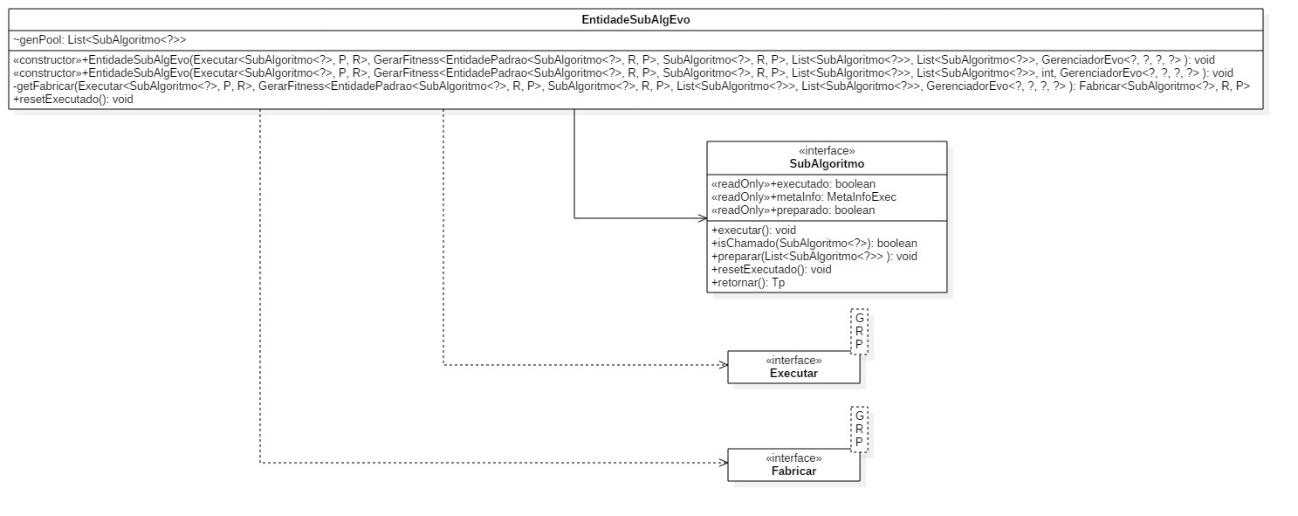


15 Classe: Coordenada e EntidadeCaixeiro



16 Classe: Log de execução de GerenciadorCaixerito.run()

Uma vez definido que as estruturas funcionam, segue-se para o desenvolvimento de outro teste. Baseando-se nas mínimas partes divisíveis de um algoritmo, a intenção era a construção de uma sub biblioteca para as estruturas programadas do software evolutivo. A tipagem do código genético seria uma interface chamada SubAlgoritmo que seria implementada por estruturas de programação, tais como: Estrutura condicional, estruturas de repetição, estruturas de contas matemática, comparação, e de operações lógicas.



17 EntidadeSubAlgEvo

A implementação foi relativamente bem-sucedida na construção de algoritmos aleatórios, uma vez resolvido o problema da normalização, que precisava garantir que todas as estruturas recebessem os parâmetros necessários (Uma soma precisa receber pelo menos dois números). Mas não fomos além disso, a aplicação do conceito de software evolutivo na construção de algoritmos sofre dificuldades severas quando se trata de verificar a unidade de medida de adaptação o *fitness*, de reproduzir as entidades ou de causar mutações nelas.

Isso significa que, apesar de conseguirmos criar algoritmos, não conseguimos alterá-los ou cruzá-los mantendo quaisquer características desejáveis.

Qualquer alteração faz com que o funcionamento se altere completamente. E não é possível um *crossover* eficiente, no que se trata de aproveitar qualidades positivas dos genitores. Pelo o menos não reaproveitando qualquer estrutura de *crossover* descrita por Zuben (2000).

A intenção inicial era uma aplicação semelhante a relativa aos sub algoritmos na construção de comandos SQL de seleção, para as palavras. Mas os mesmos problemas seriam aplicáveis, não se pode fazer qualquer mudança real em uma seleção sem alterar completamente o resultado, perdendo completamente quaisquer qualidades que se pudesse ter anteriormente, em decorrência de todos os genes, nessa análise, se referirem à uma mesma característica. Também não se pode mesclar duas seleções com simplicidade, as características vantajosas seriam perdidas com qualquer mistura.

Nesse caso, seria mais efetivo uma análise mais direcionada ao problema, perdendo a capacidade de adaptar o que se desenvolveu na resolução de outros problemas.

Essa análise mais direcionada poderia se dar com a construção de uma entidade evolutiva com mais de um método, que executaria diferencialmente, mas cada um tendo uma finalidade simples e bem definida. E resolver cada método com um único objeto, que seria a tipagem genética. Por exemplo, se a intenção fosse a construção de entidades para servirem de inimigos do usuário em um jogo. Para cada funcionalidade que o inimigo pudesse executar, haveria um gene responsável. Um que o faria se mover, um que o faria atirar, e assim por diante.

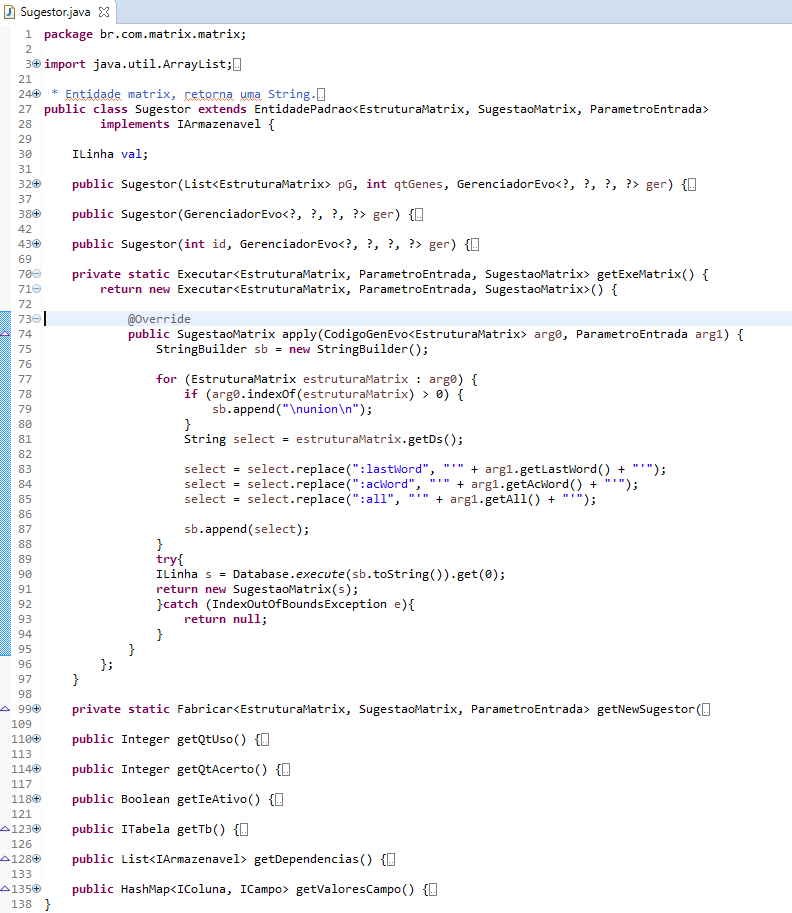
Nessa análise, um gene poderia inclusive utilizar do funcionamento de outro, como quando mover o boneco quando ele atira. Isso poderia produzir efeitos desejáveis, mas exigiria um grande cadastro de possibilidades de resolução para cada pequeno problema. Seria preciso vários tipos de movimentação e de tiro, no exemplo do jogo.

4 IMPLEMENTAÇÃO

A análise final, dada ao problema, foi a opção mais direcionada. A implementação dada à entidade tem como tipagem genética estruturas de seleção SQL, à serem cadastradas com dada escala, sem limitação declarada para a quantidade de genes.

A execução dessas entidades dá-se enviando a concatenação das estruturas para o banco, em forma de seleção. O resultado considerado é sempre o primeiro, de forma que a ordenação de interna cada gene (o order by do SQL) é extremamente impactante, bem como a ordenação das diferentes estruturas dentro da entidade.

A ordenação dos genes presentes é a alteração significativa dentro do sugestor, pois apenas o primeiro registro retornado, executando todos os selects na sequencia cadastrada no gene, é considerado. A forma vantajosa de aproveitar essa característica é que as mutações façam com que as estruturas mais abrangentes (que sempre retornam valores) estejam mais em baixo na pilha de execução.

18 Execução do sugestor

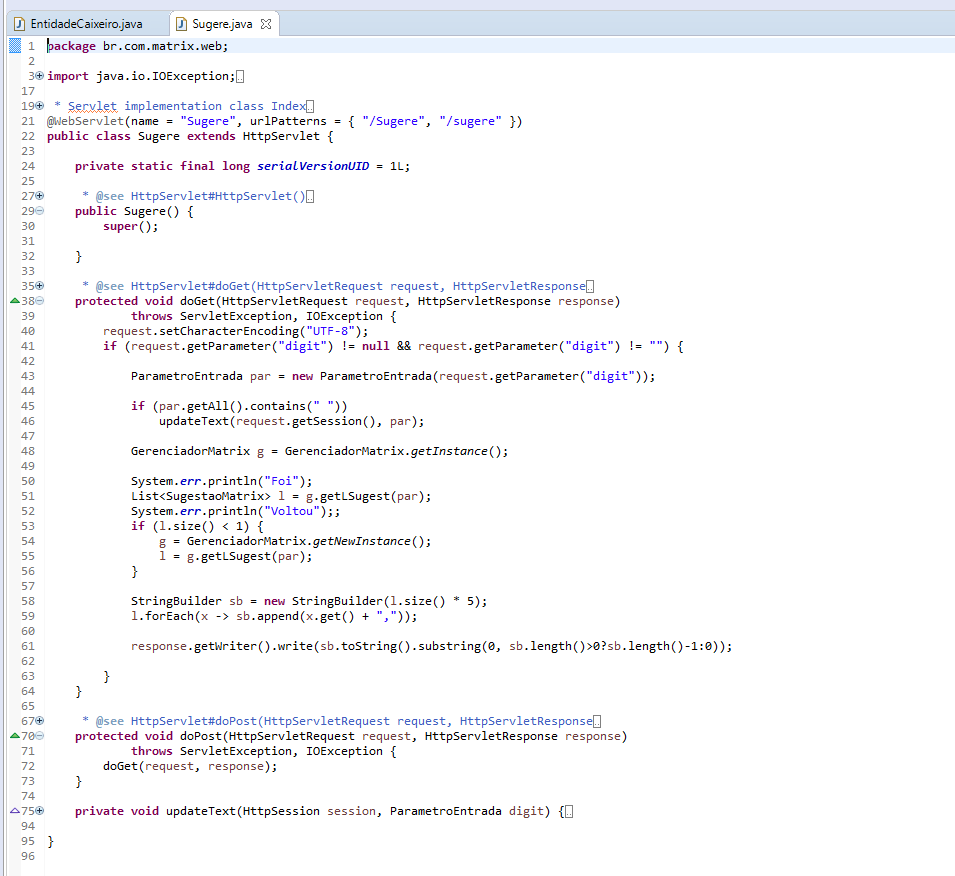
O cálculo do *fitness*, é dado com base na quantidade de sugestões que foram feitas e na quantidade de sugestões que foram aceitas.

A princípio, essa análise não valoriza muito a utilização do software evolutivo em si, uma vez que a quantidade se estruturas cadastradas será limitada. Isso é, seria possível para o programador escolher quais dessas estruturas são mais eficientes e concatena-las uma vez só. Seria o equivalente, no problema do caixeiro viajante, à um ser humano olhar no mapa e verificar a rota mais rápida.

No entanto essa análise ainda preserva algumas características desejáveis por utilizar algoritmos evolutivos. Como a adaptação à erros comuns de dado usuário, como por exemplo, adicionar, frequentemente, uma letra a mais, ou errar uma tecla.

Ainda que essas características adaptativas advindas da aplicação evolutiva na busca da melhor solução possam ser, potencialmente, resolvidas com o desenvolvimento convencional de softwares, alcança-se os objetivos estipulados no início do desenvolvimento desse trabalho.

Para integrar a Biblioteca com a parte web foram usados os Servlets, que nada mais são do que classes que fazem a conexão entre o código e o Html, com um Servlet você pode redirecionar o usuário para uma página Html sua, e passar informações a ela, que podem ser tanto objetos, quanto simples palavras. É possível também fazer requisições assíncronas aos Servlets, ou seja, requisições assíncronas, que permitem ao usuário fazer outra coisa enquanto um é pedido enviado ao Servlet por informações.



19 Servlet de sugestão.

Usamos essa tecnologia para fazer com que o autocomplete seja atualizado continuamente conforme o usuário digita algo em um campo de texto, após qualquer tecla de texto ser pressionada é enviado um Ajax com a informação do campo de texto ao Servlet, que vai separar o texto em palavras, frases e pontuações, para inseri-las no banco, logo após manda as informações resultantes para a biblioteca evolutiva, e o Servlet retorna uma lista com as palavras dadas pela biblioteca para o Html, lá a informação é pega pelo javascript e transformada em um UI de autocomplete para então mostrar à lista ao usuário.

5 CONCLUSÃO

Considerando o que foi dito, não é difícil afirmar que algoritmos evolutivos representam uma forte metodologia de desenvolvimento para um largo campo de problemas. No entanto, também ficou claro em algumas etapas do projeto que a análise mais focada na resolução do problema, e não na busca de uma metodologia de resolução, é mais efetiva.

Isso, claro, é decorrente da forma que desenvolvemos as soluções a que nos propusemos. Seria possível desenvolver uma aplicação utilizando software evolutivo na criação de sub algoritmos de forma funcional se tratados de formas menos abrangentes.

Quanto à alimentação de um autocomplete com o uso de software evolutivo, não se trata da solução mais interessante para o problema proposto. Uma vez que, como descrito, um ser humano seria capaz de identificar as estruturas que produziriam os melhores resultados, perde-se quaisquer vantagens no uso da metodologia que foi empregada. A não ser pelo conforto do usuário em reutilizar as estruturas que ele está habituado.

5.1 IMPLEMENTAÇÕES FUTURAS

Não foram implementadas as estruturas para o usuário simulado, em decorrência das conclusões anteriores de que a análise dada ao problema proposto não favoreceria uma solução eficiente. Sendo mais vantajoso cadastrar apenas uma estrutura de seleção.

Outra análise possível do problema proposto, também fazendo uso de conceitos do software evolutivo, como descrita anteriormente, é o uso de entidades (soluções possíveis) cujo código genético é não vetorial. Essa análise permitiria um intermédio do que foi desenvolvido nesse trabalho como subalgoritmos e da implementação final, aproveitando características positivas dos dois. Essas características se estendem à facilidade na implementação, e maior aproveitamento dos conceitos evolutivos.

6 REFERÊNCIAS

CAELUM. *Java e Orientação a Objetos* *2005*. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/>. Acesso em: 25 de jun. 2016.

COMPUTER HOPE. *Autocomplete 2013.* Disponível em: <<http://www.computerhope.com/jargon/a/autocomp.htm>>. Acesso em: 05 jun. 2016.

DATE, C. J. *Na Introduction to Database Systems 8e 2004.* Disponível em:

<https://robot.bolink.org/ebooks/An%20Introduction%20to%20Database%20Systems%208e%20By%20C%20J%20Date.pdf>. Acesso em: 11 jun. 2016

DEVMEDIA. *Programação Orientada a Objetos com Java 2015*. Disponível em:<Http://www.devmedia.com.br/programacao-orientada-a-objetos-com-java-easy-java-magazine-1/18449> />. Acesso em: 25 jun. 2016.

EIS, Diego. *O que é HTML básico 2011 Tableless*. Disponível em: <http://tableless.com.br/o-que-html-basico/>. Acesso em: 2 set. 2016.

ELMASRI, Ramez e NAVATHE, Shamkant B. *Fundamentals of Database Systems* 2011. Disponível em:

<http://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive\_exam/Database\_Systems.pdf>. Acesso em : 12 jun. 2016

GANGORA, Angela. *O que é inteligência artificial?* 2007. Disponível em: <www.egov.ufsc.br/portal/conteudo/o-que-e-inteligencia-artificial> Acesso em: 5 jun. 2016.

GUDWIN, Ricardo R. *Introdução a Linguagem UML.* Disponível em: <http://www.dca.fee.unicamp.br/~gudwin/ftp/ea976/Estruturais2010.pdf>. Acesso em: 08 jun. 2016.

GUIMARÃES, Isabelle, SANTOS, Carlos. *Apostila de Introdução ao CSS* 2008. Disponível em:

<https://www.telecom.uff.br/pet/petws/downloads/tutoriais/css/css2k80912.pdf>. Acesso em: 10 jun. 2016.

NERY, Michelle*. Programação Orientada a Objetos.* 2016. Disponível em: <docplayer.com.br/15805415-Linguagem-de-programacao-java-tecnico-em-informatica-professora-michelle-nery>. Acesso em: 10 jun. 2016.

NOVAES, Rafael. *O que é e para que serve IDE?* 2014. Disponível em: <http://www.psafe.com/blog/o-que-serve-ide/>. Acesso em: 06 de jun. 2016.

O.K. Takai; I.C.Italiano; J.E. Ferreira. *Introdução a Banco De Dados.* 2005. Disponível em: <https://www.ime.usp.br/~jef/apostila.pdf>. Acessado em: 8 jun. 2016.

PETER, Jr. *Introdução ao Java*. 1999. Disponível em: <www.ime.uerj.br/~alexszt/monitores/jeane/downloads/JavaPeter.pdf>. Acesso em: 9 jun. 2016

ZUBEN, Fernando. *Computação Evolutiva: Uma Abordagem Pragmática*. 2000.

Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/tutorial/tutorialEC.pdf>. Acesso em: 5 jun. 2016.