UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Alcides Gonçalves Lopes Junior

RACIOCÍNIO BASEADO EM CASOS EM SIMULAÇÃO

Santa Maria, RS
2016

Alcides Gonçalves Lopes Junior

RACIOCÍNIO BASEADO EM CASOS EM SIMULAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Luís Alvaro de Lima Silva

Trabalho de Graduação Nº 422
Santa Maria, RS
2016

Universidade Federal de Santa Maria
Centro de Tecnologia
Graduação em Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

RACIOCÍNIO BASEADO EM CASOS EM SIMULAÇÃO

elaborado por
Alcides Gonçalves Lopes Junior

como requisito parcial para obtenção do grau de
Graduando em Ciência da Computação

COMISSÃO EXAMINADORA:

Luís Álvaro de Lima Silva, Dr.
(Presidente/Orientador)

Edison Pignaton de Freitas, Dr. (UFRGS)

Mateus Beck Rutzig, Dr. (UFSM)

Santa Maria, 15 de dezembro de 2016.

RESUMO

RACIOCÍNIO BASEADO EM CASOS EM SIMULAÇÃO

AUTOR: Alcides Gonçalves Lopes Junior
ORIENTADOR: Luís Alvaro de Lima Silva

O emprego de sistemas de simulação para o treinamento de pessoas é fundamental em diferentes áreas. Técnicas de Inteligência Artificial (IA) podem ser exploradas no projeto e implementação de recursos inteligentes capazes de prover comportamento mais realístico e desafiador aos diferentes elementos que compõem estes sistemas de simulação. Dentre estas técnicas, Raciocínio Baseado em Casos (Case-Based Reasoning (CBR)) permite implementar comportamentos inteligentes dinâmicos e adaptáveis, em contraste com outras técnicas comumente utilizadas no desenvolvimento de jogos de computador, como Raciocínio Baseado em Regras. Embora técnicas de *CBR* estejam sendo exploradas no desenvolvimento de jogos de computador, essas técnicas ainda são poucos exploradas no contexto de simulação. Neste trabalho, um *framework* de desenvolvimento de sistemas *CBR* para ser utilizado na construção de simuladores na engine gráfica Unity foi planejado e implementado. Entre outras características, este *framework* permite construir uma base de experiências (ou casos), onde cada caso é formado por propriedades típicas de um cenário de simulação. O *framework* também disponibiliza diferentes medidas de similaridade para serem aplicadas na solução de consultas. A partir desta memória de experiências de solução de problemas, o *framework* permite implementar formas de recuperar casos similares para apoiar a solução de problemas dados. Para avaliar o *framework* desenvolvido, uma aplicação sobre o municiamento de uma bateria de artilharia foi projetada e implementada. Nessa aplicação, uma base de casos com 1000 (mil) diferentes casos foi construída e, a partir deles, testes envolvendo a solução de problemas de municiamento foram realizados, obtendo uma acurácia associada aos resultados obtidos superior a 60%.

Palavras-chave: Raciocínio Baseado em Casos. Simulação. RBC.

ABSTRACT

CASE-BASED REASONING ON SIMULATION

AUTHOR: Alcides Gonçalves Lopes Junior
ADVISOR: Luís Alvaro de Lima Silva

The use of simulation systems to train people is fundamental in different areas. Artificial Intelligence (AI) techniques can be exploited in the design and implementation of intelligent resources to provide more realistic and challenging behaviors to the different elements that make up these simulation systems. Among these techniques, Case-Based Reasoning (CBR) allows the implementation of dynamic and adaptive intelligent behaviors, in contrast to other techniques commonly used in the development of computer games, such as Rules Based Reasoning. Although *CBR* techniques are being exploited in the development of computer games, these techniques are still few exploited in the context of simulation. In this work, a *CBR* system development *framework* to be used in the construction of simulators in the Unity graphics engine was plan and implemented. Among other characteristics, this *framework* allows to build a case base, where each case is formed by typical features of a simulation scenario. The *framework* also provides different similarity measures to be applied in the new queries. From this memory of problem solving experiences, the *framework* allows to implement ways to recover similar cases to support the solution of problems. To evaluate the developed *framework*, an application about the choice of ammunition for an artillery battery was design and implemented. In this application, was constructed a case base with 1000 (one thousand) different cases and, from them, tests involving choice of ammunition were performed, obtaining an accuracy over of 60%.

Keywords: Case-Based Reasoning. Simulation. *CBR*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CBR	*Case-Based Reasoning*
IA	Inteligências Artificial
JSON	*JavaScript Object Notation*

SUMÁRIO

1 INTRODUÇÃO

O emprego de sistemas de simulação para o treinamento de pessoas é fundamental em diferentes áreas. Entre muitas aplicações no contexto militar, tais sistemas estão associados a capacitação de pessoal para a utilização mais segura dos equipamentos, a redução de gastos provenientes do incorreto emprego destes equipamento, a execução contínua e reprodução de exercícios de adestramento, a realização sistemática de análises pós-ação, a experimentação e desenvolvimento de novas doutrinas, etc. Em diferentes aplicações, atividades de treinamento e simulação podem ser desenvolvidos de forma integrada em ambientes de jogos de computador. Isso resulta na elaboração do conceito de jogos sérios (MICHAEL; CHEN, 2005), os quais são amplamente empregados em sistemas com fins educacionais em diferentes domínios de aplicação. Diferente de jogos de computador convencionais, contudo, jogos sérios não são fortemente voltados ao entretenimento embora existam aspectos de entretenimento que usuários também esperam usufruir nestes sistemas.

Técnicas de Inteligência Artificial (IA) podem ser exploradas no projeto e implementação de recursos inteligentes capazes de prover comportamento mais realístico a elementos de sistemas de simulação (LOPES; BIDARRA, 2011). Tanto em jogos quanto em simuladores, normalmente, o comportamento de unidades simuladas é projetado e implementado por meio de uma máquina de estados. Para isso, é utilizado algum tipo de raciocínio baseado em regras (LOPES; BIDARRA, 2011; TAMBE et al., 1995; JONES et al., 1999; EVERTSZ et al., 2007). Porém, um jogo ou simulador com um número reduzido de regras de comportamento pode se tornar repetitivo para usuários (BORCK et al., 2015; RAM; ONTAÑÓN; MEHTA, 2007). Mesmo que o número de regras aumente, quanto mais realista o comportamento dos objetos simulados deve ser, maior o número de regras que podem ser necessárias na implementação. Um grande número de regras prescritivas e estáticas de comportamento também acaba tornando o jogo ou simulador cada vez mais complexo de ser adaptado. Em contrapartida, outras técnicas inteligentes são mais dinâmicas no apoio a construção de recursos de simulação voltados para apoiar a execução de tarefas de tomada de decisão, como o Raciocínio Baseado em Casos (Case-Based Reasoning - *CBR*) (AAMODT; PLAZA, 1994; MÁNTARAS et al., 2005).

Raciocínio baseado em casos é um paradigma para a resolução de problemas fundamentado no reuso e, se necessário, adaptação do conhecimento aprendido no passado na forma de casos, os quais são utilizados para solucionar novos problemas (AAMODT; PLAZA, 1994; MÁNTARAS et al., 2005).

Em geral, sistemas baseados em técnicas de *CBR* apresenta algumas vantagens sobre sistemas baseados em regras, como: maior agilidade para desenvolvimento, capacidade de aprendizado durante sua utilização e manutenção mais simples. Porém, técnicas de *CBR* ainda são pouco exploradas na implementação de comportamentos inteligentes associados a objetos simulados em sistemas de simulação. Contudo, um domínio de aplicação semelhante a construção de

simuladores é a área de jogos de computador, onde exemplos de utilização de técnicas de *CBR* na solução de diferentes problemas podem ser encontrados. Jogos, assim como simuladores, necessitam ser desafiadores e realistas para trazer uma melhor experiência ao usuário. Neste caso, jogos do gênero Real-Time Strategy (RTS) são o domínio onde mais podemos encontrar aplicações de *CBR*. Nestes jogos, recursos automatizados baseados em técnicas de *CBR* são explorados, os quais fazem uso de experiências passadas na construção de planos para solucionar problemas específicos dos jogos (por exemplo, vencer uma partida, construir uma força para atacar um oponente) (CADENA; GARRIDO, 2011; ONTAÑÓN et al., 2010).

Em ambientes de simulação militares, simuladores virtuais e construtivos (descritos mais a frente) são extensivamente utilizados para treinamento (FLETCHER, 2009; HODSON; HILL, 2013). Na utilização de simuladores virtuais, usuários utilizam o seu conhecimento para a resolução de problemas presentes no cenário de simulação. O problema é que esse conhecimento utilizado para solucionar problemas não está sendo sistematicamente capturado por estes sistemas de simulação. Isso indica que tais experiências de solução de problemas obtidas em ambientes de simulação virtuais estão sendo completamente perdidas. Para reverter essa situação, tais experiências com simulações poderiam ser armazenadas para assim serem reutilizadas na implementação de melhores recursos inteligentes nestes sistemas de simulação. Neste contexto, este trabalho explora a utilização de *CBR* no desenvolvimento de sistemas de simulação. Para isso, o trabalho descreve o projeto, implementação e teste de um *framework* genérico de Raciocínio Baseado em Casos para dar apoio a decisão em tarefas de simulação desenvolvidas na engine gráfica Unity (TECHNOLOGIES, 2016). Devido ao fato da Unity - uma engine gráfica que está em grande expansão no mercado de desenvolvimento de jogos - não ter nenhuma solução nativa para o reuso de experiências de simulação, este trabalho inova por implementar esse *framework* de *CBR* neste ambiente. Como o *framework* de *CBR* foi desenvolvido no próprio ambiente Unity, é garantida a compatibilidade deste *framework* de *CBR* com aplicações de simulação desenvolvidas nesta plataforma. Para testar o *framework* de *CBR* proposto neste trabalho em um problema de aplicação real, um problema de municiamento de uma bateria de artilharia foi analisado e implementado. Em geral, esse problema de aplicação é utilizado como estudo de caso para a validação do *framework* de *CBR* proposto neste TCC.

No Capítulo 2 é apresentado o referencial teórico utilizado para o desenvolvimento do *framework* de *CBR*, com ênfase em simulação e *CBR*. Além disso, também são apresentados alguns trabalhos relacionados a este *framework*. No Capítulo 3 é apresentado os principais recursos da engine gráfica Unity para a criação de jogos e simuladores. Ainda no Capítulo 3, é descrito o *framework* de *CBR* desenvolvido e suas principais funcionalidades e capacidades. No Capítulo 4 é descrita a criação de uma aplicação que utiliza o *framework* de *CBR* desenvolvido. No Capítulo 5 são mostradas as conclusões a respeito do *framework* de *CBR* desenvolvido e futuros trabalhos para o mesmo.

2 REVISÃO DE CONCEITOS

Este capítulo descreve os principais conceitos sobre simulação e *CBR*. A seção sobre simulação revisa tipos de simulação e principais utilizações de simuladores. A seção de *CBR* é descreve o ciclo de *CBR* proposto por Aamodt e Plaza (1994), onde as etapas de recuperação, reutilização, revisão e retenção são descritas.

2.1 SIMULAÇÃO

Simulação pode ser descrita como uma imitação fiel, a qual é fundamentalmente baseada em características de uma situação real, ou em partes de situações reais. De acordo com Fletcher (2009), organizações militares contam com a educação e o treinamento para preparar indivíduos a atuar em situações extremamente difíceis com alto nível de eficiência. Para isso ser possível, as organizações militares utilizam de tecnologias para maximizar a eficiência e eficácia dos indivíduos nas suas tarefas. Dentre essas tecnologias existe a instrução baseada em simulação.

Simulação é dividida conforme o grau/nível de interação de pessoas/usuários com os simuladores (HODSON; HILL, 2013). Essa organização é principalmente dividida em três grandes classes (CAYIRCI, 2009; FLETCHER, 2009; HODSON; HILL, 2013):

- **Simulação Viva:** caracteriza-se por pessoas reais operando sistemas reais. Por exemplo, um piloto de uma aeronave real atirando em alvos físicos;
- **Simulação Construtiva:** caracteriza-se por pessoas simuladas operando sistemas simulados. Por exemplo, uma exibição de dois exércitos (ou unidades maiores) se enfrentando em um cenário de confronto representado em um simulador.
- **Simulação Virtual:** por pessoas reais operando sistemas simulados. Por exemplo, um aluno aprendendo a pilotar uma aeronave por meio de um simulador;

Categorizar sistemas de simulação em alguma dessas categorias pode ser problemático uma vez que não há uma distinção clara entre elas. Assim como descrito em Hodson e Hill (2013), por exemplo, o grau de participação humana em uma simulação é infinitamente variável, assim como é o grau de realismo dos equipamentos detalhados nestes simuladores. Devido a isso, sistemas de simulação são geralmente híbridos, pois há uma mistura de categorias emergindo das implementações de sistemas realizadas. Assim como ilustrado em (HEINZE et al., 2001), por exemplo, um sistema de simulação militar pode ser formado por unidades simuladas que possuem características construtivas (as quais têm um comportamento autônomo), mas também possuindo unidades com características virtuais.

Sistemas de simulação tática estão preocupados com a análise de melhores formas de organizar e empregar unidades militares, sejam essas unidades representadas por agrupamentos maiores, tais como batalhões, ou mesmo unidades menores, tais como veículos e armamentos especializados. Sistemas de simulação tática também estão preocupados com a investigação de melhores formas de emprego de técnicas para a combinação e utilização de armamento e unidades militares especializados, os quais são empregados no confronto de tropas em manobras simuladas. Em geral, tais simulações podem ser utilizadas para vários propósitos (HODSON; HILL, 2013), incluindo:

- **Análise:** busca estudar problemas complexos de planejamento, organização, logística, desenvolvimento de doutrina, avaliação da proposta de novos sistemas, entre outros. Essa atividade também é conhecida como revisão pós-ação;
- **Teste e avaliação:** ambientes virtuais de simulação são amplamente utilizados para teste. Isso se deve principalmente ao fato de que os custos de operação de ambientes reais são geralmente muito elevados;
- **Treinamento:** ambientes simulados podem ser empregados para auxiliar aqueles que precisam aprender a operar, dar suporte, manter ou interagir com sistemas reais. Em ambientes militares, o treinamento pode ser empregado para fornecer o conhecimento e habilidades necessários para executar determinadas tarefas. Assim como investigado no nosso projeto, a simulação virtual pode ser desenvolvida em um cenário de jogo de computador. Neste caso, este jogo é uma instância concreta de um jogo sério (MICHAEL; CHEN, 2005), o qual disponibiliza para alunos e instrutores militares uma imitação fiel implementada para apoiar a melhor investigação de soluções para um problema real.

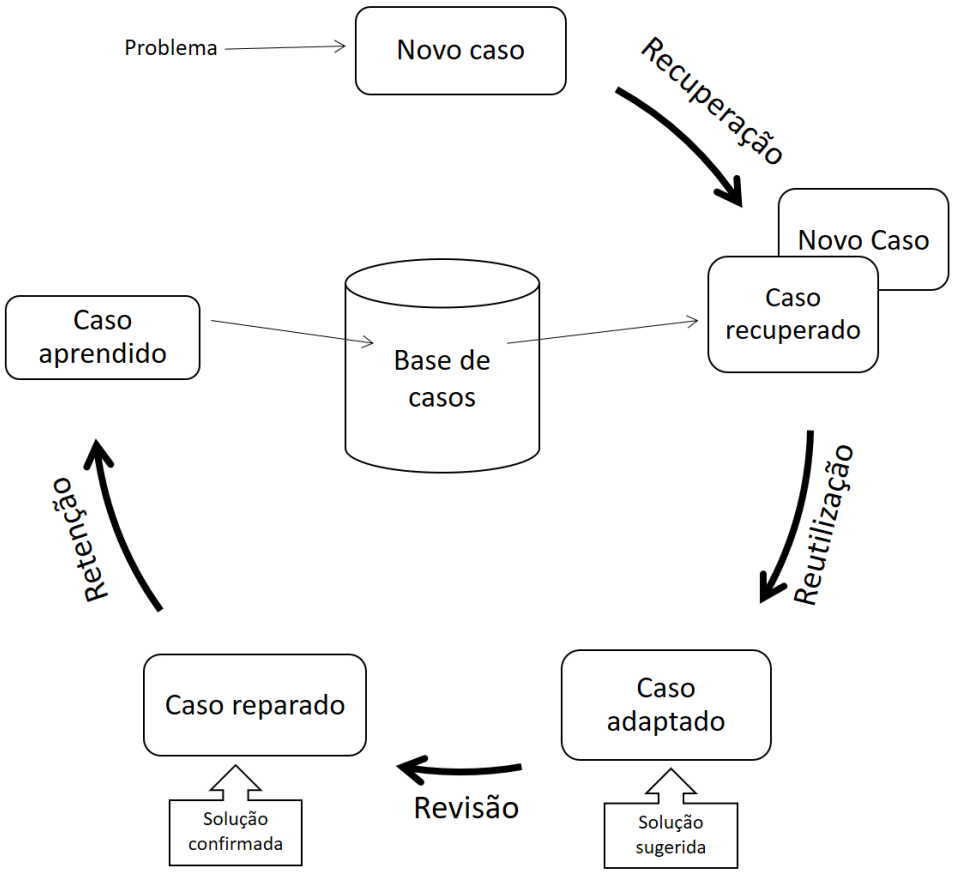
2.2 RACIOCÍNIO BASEADO EM CASOS

Raciocínio Baseado em Casos (AAMODT; PLAZA, 1994; BERGMANN, 2002; MÁNTARAS et al., 2005) é um paradigma para resolução de problemas. Este paradigma é composto por algoritmos variados, os quais são destinados a apoiar a construção de sistemas inteligentes em diferentes problemas de aplicação (MÁNTARAS; CUNNINGHAM; PERNER, 2005; WATSON, 2003) . A ideia básica deste paradigma é resolver um novo problema relembrando uma situação anterior que é similar ao problema atual sendo resolvido e, então, reutilizando a informação e conhecimento daquela experiência de solução de problemas passada na solução deste problema atual. Neste contexto, é importante salientar que sistemas de *CBR* funcionam por meio do reuso de experiências concretas de solução de problemas (ou casos) obtidas no passado. Tais experiências representam uma memória de solução de problemas. Em sistemas

CBR, tal memória, representada como uma base de casos, a qual pode capturar melhores formas (e situações que podem ser evitadas) de solução de problemas de forma explícita e reutilizável.

Sistemas *CBR* necessitam de mecanismos para recuperação casos armazenados em bases de casos. Em muitas situações, contudo, soluções gravadas em casos recuperados podem não ser diretamente reusáveis na solução de novos problemas. Para isso, pode ser necessário adaptar essas soluções passadas, visto que podem existir diferenças entre o problema corrente e os casos mais similares recuperados da base de casos. Uma vez que uma solução seja encontrada pelo sistema *CBR*, esta solução proposta ainda precisa ser validada. A ideia é que a solução encontrada pelo reuso de experiências passadas não seja difícil de ser aplicada no problema corrente. Por fim, muitos sistemas de *CBR* armazenam o conhecimento obtido durante esse ciclo de solução de problemas. Tal ciclo pode ser descrito em etapas (Figura 2.1:

Figura 2.1 – Ciclo do paradigma de Raciocínio Baseado em Casos



Fonte: Adaptado de (AAMODT; PLAZA, 1994)

- Recuperar um ou mais casos similares ao problema atual;
- Reutilizar o conhecimento presente neste(s) caso(s) para solucionar o novo problema;
- Revisar a solução proposta;

- Reter, caso necessário, a experiência aprendida para ser utilizada na solução de novos problemas.

Em geral, este ciclo não precisa ocorrer em sua totalidade em muitos sistemas *CBR*. Entre outros motivos, muitos problemas de aplicação podem ser amplamente atacados com a implementação de recursos de recuperação de casos eficientes, por exemplo. Isso indica que algumas destas etapas, tais como as envolvidas em processos de adaptação, por exemplo, podem ser desenvolvidas manualmente pelos próprios usuários de sistema *CBR*, ao invés de serem completamente automatizadas por diferentes algoritmos.

2.2.1 A Base de Casos

Um sistema de *CBR* é dependente de sua coleção de casos armazenados na base de casos. Para este sistema ser eficiente, a base de casos deve conter casos capturando diferentes problemas e formas de resolver esses problemas. Tal base de casos pode ser armazenada computacionalmente de diversas formas estruturadas ou semiestruturadas (BERGMANN; KOLODNER; PLAZA, 2005). Uma forma simples de representar casos em uma base de casos é baseada na estrutura de documentos JSON, a qual permite capturar explicitamente diferentes propriedades dessas experiências de solução de problemas.

Para encontrar casos similares na base de casos para um problema dado (descrito como uma consulta), é necessário definir quais atributos (ou propriedades) devem ser utilizados na comparação entre um caso da base de casos e o problema corrente. Estes atributos, utilizados na comparação entre casos, são denominados índices. Estes são a combinação dos atributos mais importantes do caso, os quais representam informações consideradas úteis para resolver problemas. Normalmente, esses atributos permitem distinguir um caso dado de outros. Entretanto, a escolha de quais atributos que devem ser indexados não é uma tarefa simples. Em muitos problemas, essa tarefa envolve a investigação e identificação de atributos relevantes para a tomada de uma decisão no problema de aplicação sendo tratado. Uma vez escolhido um conjunto de propriedades que permitem caracterizar um caso, a representação de um caso descreve as características do problema e como ele foi resolvido. Em muitos sistemas *CBR*, isso é representado em um par (problema, solução):

- **Descrição do problema:** descreve as características do ambiente no momento que o caso ocorreu e as restrições associadas a este problema;
- **Descrição da solução:** descreve os conceitos ou objetos usados para resolver o problema, considerando as restrições especificadas e a descrição da situação. Esta solução pode ser uma ação, um conjunto de procedimentos, uma classificação, etc;

2.2.2 Recuperação

A primeira etapa do ciclo de *CBR* é a recuperação. Esta tem como objetivo encontrar um ou mais casos que contenham uma solução útil para solucionar o problema atual. Nesta etapa, após a entrada de um novo caso como consulta, o sistema *CBR* faz o *matching* desse novo caso com cada caso da base de casos. Para isso, um algoritmo de similaridade é utilizado, buscando um ou mais casos que possuam semelhança ao novo caso. Em geral, se um novo caso tem semelhanças significativas entre a descrição de seu problema e a descrição do problema contida no caso passado, espera-se que o novo caso e o caso passado apresentem soluções semelhantes. Devido a essas semelhanças, a solução do caso passado pode vir a ser relevante, e consequentemente reusada, na solução do problema atual.

Uma medida de similaridade é a formalização de um modelo matemático a ser utilizado na implementação de mecanismos de recuperação em sistemas *CBR*. Esta medida pode ser formulada como uma medida de distância entre casos. Para o cálculo desta medida de similaridade, dois aspectos são relevantes (MÁNTARAS et al., 2005): i) a similaridade local entre atributos individuais de um problema e ii) a similaridade global entre casos.

A similaridade local representa a similaridade entre cada atributo do caso tomado individualmente. Para cada tipo de atributo, uma medida de similaridade local é empregada (BERGMANN, 2002). Para medir a similaridade local entre atributos numéricos, por exemplo, é possível empregar uma função linear (Equação 2.1). Nesta função, a ideia é que a similaridade deva crescer com o decréscimo da distância entre os dois valores de atributos sendo comparados, onde esta medida é ponderada pelo tamanho do intervalo assumido pelo domínio dos valores possíveis para o atributo analisado.

$$SimLocal(a_i, b_i) = \frac{\mid b_i - a_i \mid}{maxValue - minValue} \tag{2.1}$$

A similaridade global representa a similaridade entre dois casos tomados como um todo. Uma das técnicas mais populares para o cálculo dessa similaridade é baseada no emprego de uma equação de distância Euclidiana (Equação 2.2), a qual permite medir a distância entre o caso da consulta e cada um dos casos da base de casos tomado individualmente. A distância Euclidiana é calculada por meio da raiz quadrada do somatório das similaridades locais de cada propriedade indexada de um caso, onde o resultado é elevado ao quadrado e multiplicado por um peso w_i. Este peso é referente a importância do índice i na análise de similaridade. Neste caso, certas propriedades de um problema podem ser mais relevantes do que outras na comparação entre dois casos. Geralmente, a similaridade entre dois casos é representada por um número real no intervalo [0,1], onde o valor 1 indica a total igualdade entre os casos. Por fim, o valor desse somatório deve ser normalizado pelo somatório dos pesos w dos atributos analisados, os quais descrevem a importância relativa de cada atributo do problema.

$$SimGlobal(c,q) = \sqrt{\frac{\sum_{i=0}^{n} SimLocal(c_i,q_i)^2 * w_i}{\sum_{i=0}^{n} w_i}} \quad (2.2)$$

O *matching* de casos pode ser definido como o processo de comparar as propriedades do problema atual com as propriedades de um problema gravado em um caso passado. Isso consiste em aplicar uma medida de similaridade sucessivamente a cada caso da base de casos e, após isso, ordenar os casos de acordo com sua medida de similaridade global. Como resultado deste tipo de consulta, os casos mais similares ao caso da consulta são retornados.

2.2.3 Reutilização

Após recuperados um ou mais casos pela etapa anterior (Capítulo 2.2.2) e selecionado o mais útil para a solução do novo caso, a segunda etapa do ciclo de *CBR*, a Reutilização, é iniciada. Nesta etapa, dois diferentes métodos podem ser empregados para utilizar a solução do caso mais útil recuperado (AAMODT; PLAZA, 1994; MÁNTARAS et al., 2005): copiar a solução ou adaptar a solução.

Copiar a solução do caso recuperado é a mais simples. Baseia-se em transferir a solução do caso recuperado para a solução do novo caso. Porém, alguns sistemas necessitam levar em conta as diferenças entre o caso passado e o caso presente (AAMODT; PLAZA, 1994). Com isso, a solução precisa ser adaptada antes de ser transferida para o novo caso.

Adaptar uma solução já é uma tarefa mais complexa do que simplesmente copiá-la. Esta busca satisfazer os requisitos e restrições do problema atual sendo atacado. Uma abordagem utilizada para adaptar soluções é observar como o problema do caso passado foi resolvido (AAMODT; PLAZA, 1994) por meio das justificativas do por que determinada ação foi tomada, dos objetivos a serem atingidos, etc.

Em muitos problemas de aplicação, os próprios usuários do sistema *CBR* adaptam manualmente (ou de forma semiautomática) a solução do caso passado para atender o problema atual.

2.2.4 Revisão

Quando a etapa de Reutilização (Capítulo 2.2.3) gera uma solução incorreta, a etapa de revisão é utilizada para avaliar e reparar a solução proposta. Se a solução for considerada correta, ela pode ser aceita. Caso contrário, a solução deve ser reparada, o que normalmente envolve utilizar conhecimento específico sobre o problema sendo resolvido. Na prática, esta etapa de *CBR* indica que a solução gerada pelo sistema pode sofrer algum tipo de intervenção humana visto que o sistema pode eventualmente gerar soluções adaptadas que raramente acontecem na

realidade.

2.2.5 Retenção

A retenção é responsável pelo armazenamento de um novo caso resolvido na base de casos. Esta etapa somente acontece se usuários consideram relevante armazenar o caso resolvido, o que implica em reter essas novas experiências de solução de problemas para futuro reuso. Na maioria das vezes, um novo caso somente é armazenado na base de casos caso ele permita o sistema *CBR* resolver um novo tipo de problema no futuro. Nesta situação, o novo caso solucionado com o auxílio do sistema *CBR* descreve uma experiência de solução de problemas nova, a qual deve ser mantida para análise e reuso futuros.

2.3 TRABALHOS RELACIONADOS

O desenvolvimento de forças geradas por computador é uma área de desenvolvimento de software comumente explorada no processo de construção de simuladores para problemas de aplicação militar. Utilizando técnicas inteligentes variadas, forças geradas por computador são utilizadas na solução de problemas voltados para promover diferentes tipos de treinamento militar, onde unidades autônomas sendo simuladas devem ter um comportamento fiel a uma doutrina e tática (LAIRD; VANLENT, 2001). Tambe et al. (1995) e Jones et al. (1999) utilizaram, para a geração de forças geradas por computador, um sistema conhecido como TacAir-Soar. Esse é um sistema baseado em regras, criado a partir da arquitetura Soar (TAMBE et al., 1995). Este sistema é utilizado em exercícios militares de simulados (TAMBE et al., 1995; JONES et al., 1999), mais especificamente em um simulador de combate aéreo.

De acordo com Jones et al. (1999), Tambe et al. (1995), pilotos autônomos de aeronaves presentes no cenário de simulação utilizam comportamentos implementados no sistema TacAir-Soar para executar ações de combate aéreo realistas em cenários de simulação. Neste caso, situações de combate são interpretadas, ações são propostas, selecionadas e aplicadas por um piloto autônomo presente na simulação, respeitando a doutrina militar imposta pelas regras implementadas no sistema TacAir-Soar (TAMBE et al., 1995).

Além de sistemas baseados em regras, outras técnicas inteligentes também podem ser exploradas no desenvolvimento de forças geradas por computador. Dentre elas, técnicas que focalizem o papel de experiências passadas na solução de novos problemas, assim como disponibilizado por técnicas de *CBR*. Por exemplo, Reece, McCormack e Zhang (2004) criaram um sistema *CBR* com o objetivo de facilitar a criação comportamentos inteligentes para objetos simulados. O sistema funciona da seguinte forma: o usuário cria novos comportamentos por meio da construção de uma máquina de estados em uma interface gráfica; tais comportamen-

tos são armazenados como uma solução para o caso que está sendo criado; o usuário informa a descrição do problema que esta solução é aplicada, assim completando a construção de um caso; o usuário pode então realizar consultas para recuperar comportamentos já existentes em uma base de casos e decidir se pode utilizá-los ou deve adaptá-los manualmente na solução de um novo problema envolvendo o objeto sendo simulado.

No sistema *CBR* descrito por Reece, McCormack e Zhang (2004), usuários interferem manualmente na criação do caso, bem como na decisão se o caso retornado para uma consulta pode ser reutilizado ou não. Usuários também são responsáveis por adaptar soluções disponíveis em casos recuperados, caso não seja possível utilizar essas soluções sem que alguns ajustes sejam realizados. Porém, existem sistemas *CBR* completamente autônomos, onde não existe interferência usuários no processo de tomada de decisão. Por exemplo, Borck et al. (2015) desenvolveram um sistema inteligente autônomo de *CBR*, como parte de um simulador de combate aéreo. Entre outras características, este sistema é capaz de predizer a ação que agentes aliados devem tomar de acordo com o comportamento dos agentes inimigos. Em resumo, técnicas de *CBR* são utilizadas na implementação de tarefas de predição, e consequente planejamento de ações baseadas nas predições realizadas.

Na construção de simuladores, *CBR* pode ser utilizado com a justificativa de reduzir a complexidade de criação de comportamentos, visto que a ideia de *CBR* é promover a reutilização de comportamentos passados (BORCK et al., 2015; REECE; MCCORMACK; ZHANG, 2004). Porém, *CBR* também pode apresenta algumas complexidades, como: o número de casos presentes na base de casos pode não ser suficiente para resolver os problemas do domínio de aplicação alvo, os atributos da aplicação escolhidos e indexados e a relevância que cada atributo tem em consultas realizadas devem ser conhecidos para assim reduzir a possibilidade de recuperar casos incorretos para solucionar novos problemas propostos, entre outros. Em geral, a aplicação de *CBR* na construção de simuladores ainda é uma área pouco explorada. Contudo, existem outras áreas da computação que possuem semelhanças com a área de simulação, por exemplo, a área de jogos de computador, onde técnicas de *CBR* têm sido mais utilizadas.

Embora o processo de desenvolvimento de jogos tenha avançado muito na qualidade gráfica, animações e áudio, tais jogos ainda utilizam técnicas de IA bastante simplificadas. Isso pode ser explicado pelo fato de que, geralmente, é alocado pouco tempo de CPU para a execução de tarefas de IA (neste caso, tempo de CPU é normalmente utilizado na computação das interfaces e modelos gráficos desses jogos). Neste caso, grande parte das técnicas mais recentes de IA necessitam de maior tempo de processamento, ou seja, tempo para obter resultados satisfatórios. Isso tende a ocasionar um atraso nas ações executadas por objetos de um jogo. Isso é indesejável, visto que esse atraso pode prejudicar a experiência do usuário ao jogar o jogo (ONTAÑÓN et al., 2010). Assim, novas técnicas e métodos de IA foram desenvolvidos buscando tornar as ações dos agentes envolvidos no jogo mais realistas sem prejudicar a experiência do jogador durante o jogo. Dentre essas técnicas e métodos, existe o On-Line Case-Based Planning (ONTAÑÓN et al., 2010).

O On-Line Case-Based Planning (OLCBP) é uma extensão do paradigma de *CBR* para ser utilizado "on-line"durante um jogo, particularmente, em jogos do gênero de Estratégia em Tempo Real (Real Time Strategy - RTS). Esse gênero apresenta uma severa restrição de tempo de tomada de decisão para usuários ou agentes autônomos envolvidos no jogo. Assim, o desafio dessa extensão é planejar ações e, possivelmente, adaptá-las, respeitando a limitação de tempo de processamento e tomada de decisão impostas.

Jogos de RTS como Wargus (cópia com código aberto do jogo Warcraft II) (ONTAÑÓN et al., 2010; SUGANDH; ONTAÑÓN; RAM, 2008) e Starcraft (ERIKSSON; TORNES, 2012) são alvos da técnica OLCBP para realizar o planejamento de ações. Em geral, o objetivo dessas implementações autônomas é construir um agente autônomo que possa vencer o jogo (por exemplo, uma partida de futebol). Além do planejamento de ações e predição, *CBR* também é aplicado em tarefas de microgestão (LARA-CABRERA; COTTA; LEIVA, 2013). Por exemplo, Cadena e Garrido (2011) construíram um sistema *CBR* como parte do jogo de RTS StarCraft. Neste sistema, o problema é capturado por um conjunto de características da área de um terreno virtual analisado. Nesta área, estão presentes unidades amigas e inimigas, checkpoints, etc. Cada solução neste sistema é composta por uma sequência de ações táticas que devem ser tomadas, uma vez que o cenário do jogo tenha sido analisado. Por exemplo, atacar unidades na área, mover e atacar unidades em outra área, quais unidades utilizar, etc.

Tanto jogos quanto simuladores também necessitam ser adaptáveis aos usuários que os estão utilizando Laird e VanLent (2001), Lopes e Bidarra (2011). Neste caso, se o usuário é mais experiente no jogo, recursos de IA devem tornar o jogo mais difícil. Se o usuário for menos experiente, o jogo deve ser mais fácil. Importante, esse ajuste é realizado de forma autônoma, e não é baseado em um processo determinístico tal como um ajuste obtido pela escolha de diferentes níveis de dificuldade pré-programados em diferentes tipos de jogos.

Analisando os trabalhos apresentados neste capítulo, é possível notar a semelhança entre os desafios e objetivos presentes tanto em simulação quanto em jogos. É possível destacar a importância do realismo e adaptabilidade que simuladores e jogos devem prover aos seus usuários para garantir uma melhor experiência na sua utilização. Além disso, é possível notar que o planejamento, a predição e a microgestão (descrita nos trabalhos sobre jogos de RTS) são características que também são desejáveis de serem implementadas em Forças Geradas por Computador (descrita nos trabalhos sobre simulação).

3 METODOLOGIA E FERRAMENTAS

3.1 O AMBIENTE UNITY: ENGINE GRÁFICO PARA O DESENVOLVIMENTO DE JOGOS E SIMULADORES

A Unity (TECHNOLOGIES, 2016) é uma engine gráfica, desenvolvida pela *Unity Technologies*, para o desenvolvimento de jogos, a qual também pode ser empregada para o desenvolvimento de simuladores. Ela apresenta diversos componentes necessários para a criação de jogos, como:

- **Gráficos:** oferece um conjunto de funcionalidades que permitem criar cenários realistas para uma aplicação desenvolvida. Dentre estas funcionalidades, recursos de iluminação, materiais, sombras e partículas podem ser destacados;
- **Física:** para ter um comportamento físico convincente, um objeto deve acelerar corretamente e ser afetado por colisões, gravidade e outras forças. A Unity dispõe de componentes que tratam dessas forças automaticamente, onde é necessário configurar parâmetros para essas forças para obter uma simulação física realista;
- **Scripting:** jogos e simuladores dependem de scripts para responder as entradas de um jogador, bem como organizar eventos internos as aplicações desenvolvidas. Na Unity, tais scripts também são utilizados para controlar recursos gráficos, controlar recursos físicos e até mesmo para a criação de componentes de IA para objetos utilizados nas aplicações desenvolvidas;
- **Rede e multiplayer:** permite a criação de jogos online, provendo ferramentas que gerenciam a rede e a comunicação entre os diversos players que estão conectados a um servidor de aplicação;
- **Áudio:** jogos e simuladores também dependem da sonorização para terem um aspecto realista. A Unity contém componentes que realizam a reprodução de som, a mixagem e masterização em tempo real e a execução de efeitos predefinidos, entre outros;
- **Animação:** ontém recursos de animação que incluem animações reajustáveis, formas para mistura de animações faciais, controle das animações em tempo de execução, entre outros;
- **Interfaces de usuário:** contém componentes que são adaptáveis e podem ser combinados para a criação de interfaces de usuário para diferentes tipos de aplicações;

- **Navegação e Pathfinding:** apresenta recursos de navegação para mover, de forma autônoma, objetos dentro do cenário de um jogo ou simulador. Para isso, a Unity utiliza malhas de navegação que são criadas automaticamente com base em aspectos geométricos de uma cena/cenário. Essa malha pode ser adaptada em tempo de execução, ou seja, o itinerário a ser percorrido pelo objeto pode ser alterado dado um novo obstáculo.

3.2 RACIOCÍNIO BASEADO EM CASOS EM SIMULADORES VIRTUAIS

Por meio de técnicas de *CBR*, a captura e o reuso de experiências obtidas com a solução de problemas são funcionalidades relevantes no contexto de desenvolvimento de jogos e sistemas de simulação. Para que isso seja possível, contudo, é necessário gravar essas experiências (ou casos) em um formato que possa ser utilizado e compartilhado por sistemas de simulação. Uma vez que tais experiências estejam representadas em um formato computável, é possível:

- Utilizar técnicas de *CBR* para automaticamente realizar, de forma realista, tarefas complexas em sistemas de simulação. Neste caso, tarefas de tomada de decisões associadas a objetos simulados podem ser abordadas utilizando essas experiências;

- Utilizar técnicas de *CBR* para possibilitar a análise automática e correção de escolhas/decisões realizadas por usuários em cenários de simulação, comparando essas escolhas com as experiências passadas gravadas. Por exemplo, uma tarefa de revisão pós-ação pode ser parcialmente automatizada por um sistema que utiliza experiências concretas a respeito de escolhas realizadas em exercícios de simulação passados;

- Utilizar técnicas de *CBR* para auxiliar usuários em processos de solução de problemas, ou seja, treinar esses usuários. Neste trabalho, por exemplo, técnicas de *CBR* são utilizadas na criação de uma ferramenta para o apoio a decisão na tarefa de municiamento de baterias de artilharia;

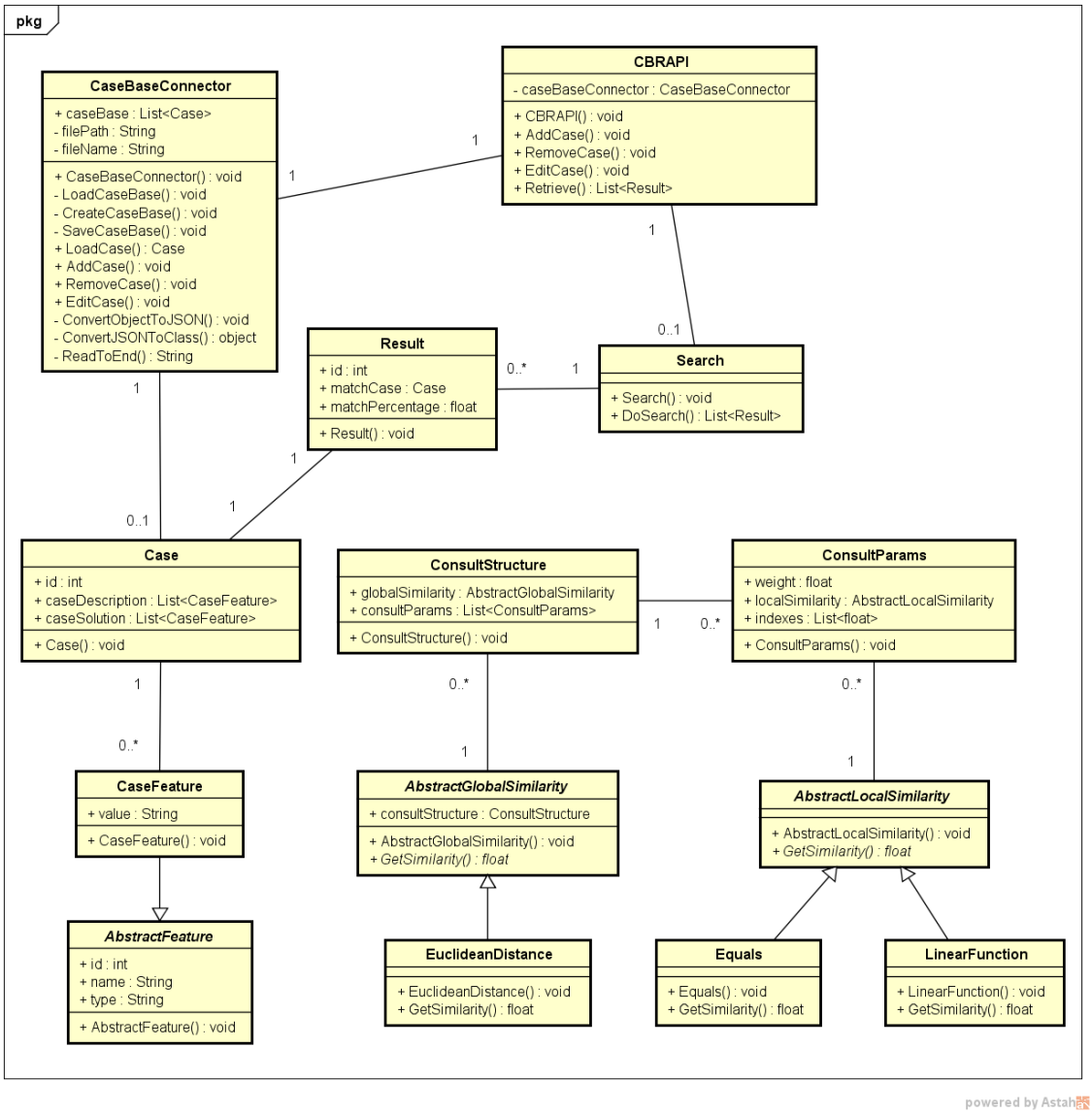
- Utilizar técnicas de *CBR*, em contraste com a exploração de técnicas que são baseadas na elicitação e representação de um grande número de regras. Em geral, a ideia é utilizar as experiências passadas para implementar processos de tomada de decisão menos preditivos/determinísticos (e mais fiéis a processos de tomada de decisões reais, os quais podem ser desenvolvidos em sistemas de simulação);

Em resumo, para que essas experiências possam ser exploradas na construção de novos recursos em simuladores, é necessário existir um *framework* de *CBR* implementado no ambiente Unity.

3.2.1 Uma biblioteca de *CBR* implementada no ambiente Unity

O objetivo principal deste *framework* de *CBR* é prover um conjunto de classes e métodos (mostrados no diagrama de classes presente na Figura 3.1)para o desenvolvimento de aplicações de *CBR* dentro da engine gráfica Unity. Em geral, as classes *CBR* desenvolvidas podem ser extendidas e adaptadas para o desenvolvimento de diferentes problemas de aplicação. Entre essas funcionalidades implementadas neste *framework* de *CBR*, é possível destacar:

Figura 3.1 – Diagrama de classes do *framework* de *CBR* desenvolvido



- **A base de casos e seu gerenciador:** no *framework* de *CBR* implementado na Unity, uma base de casos de um problema de aplicação é constituída a partir da leitura e processamento de informações armazenadas em um arquivo no formato JSON (THE. . . , 2013). Um exemplo deste arquivo é mostrado na Figura 3.2. Neste arquivo, cada item é composto por duas listas, as quais refletem a representação de um caso por um par (problema,

solução). A primeira lista é referente aos atributos presentes na descrição do problema de um caso. A segunda captura os atributos presentes na descrição da solução de um caso. Além disso, existe uma classe chamada CaseBaseConnector. Esta classe é responsável por gerenciar a base de casos, permitindo adicionar novos casos, como também remover ou editar casos já existentes. Assim como planejado no *framework* de *CBR* implementado, a classe CaseBaseConnetor só pode ser acessada por meio da classe *CBR*API. A classe *CBR*API é a principal deste *framework* de *CBR* implementado na Unity, visto que ela é utilizada no desenvolvimento de uma nova aplicação para ter acesso as funcionalidades de gerenciamento da base casos.

Figura 3.2 – Exemplo de caso, no formato JSON, presente na base de casos


```
{
    "caseBase": [
        {
            "id": 0,
            "caseDescription": [
                {
                    "id": 0,
                    "name": "Tipo do alvo",
                    "type": "System.String",
                    "value": "AlvoTipo3"
                },
                {
                    "id": 1,
                    "name": "Distancia do alvo",
                    "type": "System.Single",
                    "value": "24.557"
                },
                {
                    "id": 2,
                    "name": "Area do alvo",
                    "type": "System.Single",
                    "value": "1.25"
                },
                {...},
                {...},
                {...},
                {...},
                {...},
                {...},
                {...},
                {...},
                {...},
                {...}
            ],
            "caseSolution": [
                {
                    "id": 0,
                    "name": "Foguete Utilizado",
                    "type": "System.String",
                    "value": "FogueteTipo1"
                },
                {
                    "id": 1,
                    "name": "Numero de foguetes utilizados",
                    "type": "System.Int32",
                    "value": "51"
                },
                {...}
            ]
        }
    ]
}
```


Fonte: Arquivo JSON que representa a base de casos

- **A estrutura dos casos:** a classe Case é utilizada para representar os casos que serão utilizados para popular uma base de casos para um determinado problema de aplicação.

Ela também é utilizada para representar casos utilizados como consultas em aplicações de *CBR* desenvolvidas. Esta classe é constituída por duas listas de atributos, os quais representam as principais características de um problema (neste caso, as propriedades que são indexadas em um problema de aplicação). A primeira representa a descrição do problema e a segunda representa a descrição da solução. Os atributos destas listas se referem as informações necessárias para representar uma experiência (ou caso) em uma forma computacionalmente interpretável. Assim, essas duas listas permitem compor um caso em uma aplicação particular, o qual ver ser armazenado na base de casos. Essas listas também permitem compor casos que serão utilizados como consultas na aplicação *CBR* sendo construída;

- **As medidas de similaridade:** o *framework* de *CBR* implementado na Unity permite a criação de medidas de similaridade globais e medidas de similaridade locais. Para isso, é necessário entender a classe AbstractGlobalSimilarity ou a classe AbstractLocalSimilarity, respectivamente. Também é necessário criar métodos concretos a partir dos métodos abstratos representados nessas classes, ou seja, para a criação de novas medidas de similaridade é necessário conhecimento do usuário sobre essas medidas. Desta forma, as funções de similaridade utilizadas para computar respostas para consultas em aplicações *CBR* podem ser utilizadas. Por exemplo, o *framework* de *CBR* implementado apresenta medidas simplificadas de similaridade local desenvolvidas: uma função linear (Equação 2.1), descrita pelo Algoritmo 1, uma função que analisa se dois atributos são iguais, descrita pelo Algoritmo 2 e, por fim, uma medida de similaridade global distância Euclidiana (Equação 2.2) desenvolvida no formato do Algoritmo 3;

Algoritmo 1: LINEARFUNCTION

Entrada: $consultParams, searchCase, retrieveCase$

Saída: Valor de similaridade entre 0 e 1

1 **início**

2 $valor1 \leftarrow GetValue(searchCase, consultParams)$

3 $valor2 \leftarrow GetValue(retrieveCase, consultParams)$

4 $similarity \leftarrow 1 - \frac{|\ valor2 - valor1\ |}{maxValue - minValue}$

5 **retorna** $similarity$

6 **fim**

Algoritmo 2: EQUALS

Entrada: $consultParams, searchCase, retrieveCase$

Saída: Valor de similaridade entre 0 e 1

```
1 início
2     valor1 ← GetValue(searchCase, consultParams)
3     valor2 ← GetValue(retrieveCase, consultParams)
4     se valor1 = valor2 então
5         retorna 1
6     senão
7         retorna 0
8     fim
9 fim
```

Algoritmo 3: EUCLIDEANDISTANCE

Entrada: $searchCase, retrieveCase$

Saída: Valor de similaridade entre 0 e 1

```
1  início
2      similarity ← 0
3      sumWeights ← 0
4      para cada consultParams ∈ GetConsultStructure() faça
5          se consultParams.HasBlankFeature() então
6              continue
7          fim
8          se consultParams.similarityMeasure ≠ null então
9              similarity ← similarity +
                 GetLocalSimilarity(consultParams, searchCase, retrieveCase)^2 *
                 consultParams.weight
10             sumWeights ← sumWeights + consultParams.weight
11         fim
12     fim
13     se sumWeights = 0 então
14         retorna 0
15     fim
16     retorna sqrt(similarity / sumWeights)
17 fim
```

- **A estrutura de uma consulta:** para ser possível realizar uma consulta em uma aplicação *CBR* desenvolvida a partir da *framework CBR* proposto, é necessário criar a estrutura desta consulta. Essa estrutura é descrita pela classe CaseStructure. Esta classe é composta

por zero ou mais parâmetros de uma consulta. Cada parâmetro da consulta (representado pela classe CaseParams) é constituído por zero ou mais índices, os quais representam os atributos presentes na descrição de um caso. Isso é necessário pois em determinadas computações de similaridade, além do atributo analisado, são necessárias outras informações do caso. Nesta situação, por exemplo, além destes índices, também é necessário saber o peso (ou relevância) que o parâmetro de consulta deve ter na análise de similaridade entre dois casos, bem como a medida de similaridade local (descrita no Capítulo 2.2.2) que será utilizada;

- **O ciclo de *CBR*:** no *framework* de *CBR* implementado na Unity, apenas a etapa de recuperação do ciclo de *CBR* foi implementada. Ela pode ser executada chamando o método Retrieve da classe *CBR*API (Figura 3.1). Este método recebe como parâmetro o caso utilizado como consulta e a estrutura da consulta. Após isso, é instanciada a classe Search, a qual realiza a análise de similaridade entre o caso utilizado como consulta em relação a todos os casos disponíveis na base de casos, seguindo o Algoritmo 4. Cada caso da base de casos utilizado nesta comparação com a consulta dada é adicionado na lista de resultados, onde esse caso recuperado é associado a um valor de similaridade com o caso utilizado na consulta. Ao completar esse passo de recuperação de casos, a lista de casos recuperados e seus valores de similaridade computados pela consulta são retornados para o usuário.

Algoritmo 4: DOSEARCH

Entrada: $searchCase, caseBase, consultStructure$

Saída: Lista contendo os resultados da consulta

```
1 início
2     para cada case ∈ caseBase faça
3         similarity ← consultStructure.GetGlobalSimilarity(searchCase, case)
          result ← GetRetrievalResult(similarity, case)
          results ← AddRetrievalResult(result)
4     fim
5     retorna results
6 fim
```

4 UMA APLICAÇÃO UTILIZANDO O *FRAMEWORK* DE *CBR*

Neste capítulo, os passos utilizados para a criação de uma aplicação utilizando o *framework* de *CBR* desenvolvido são descritos. A aplicação envolve a tarefa de municiamento de baterias de artilharia. Neste caso, o objetivo desta aplicação é utilizar experiências passadas de municiamento de baterias para automaticamente determinar o melhor tipo e quantidade de munição que deve ser empregado por uma nova bateria para alvejar um determinado alvo.

4.1 PASSOS PARA CONSTRUIR UMA APLICAÇÃO DE *CBR* E REALIZAR UMA CONSULTA NESTA APLICAÇÃO

A sequência de passos que deve ser seguida para a criação de uma aplicação utilizando o *framework* de *CBR* descrito é a seguinte:

- **Passo 1 (Instanciação da classe *CBR*API):** o primeiro passo envolve instanciar a classe *CBR*API e informar, via argumento, o nome do arquivo, no formato JSON, que representa a base de casos (ou representará, caso essa base ainda não tenha sido criada). É de extrema importância que cada item (ou caso) deste arquivo esteja no mesmo formato da estrutura do caso criado no *framework* de *CBR*, ou seja, apresentar o mesmo número de atributos com seus respectivos tipos. Se isso não acontecer, ocorrerá erros na leitura do arquivo. Por exemplo, a Figura 4.1 demonstra como é realizada a instanciação da classe *CBR*API, informando o arquivo JSON que contém a base de casos (CaseBase.json)

Figura 4.1 – Trecho de código referente instanciação do *framework* de *CBR* com o nome de base de casos

```
CBRAPI cbrAPI = new CBRAPI("CaseBase.json");
```

Fonte: Código do *framework* de *CBR*

- **Passo 2 (Criação da estrutura de um caso):** o segundo passo para a criação de uma aplicação de *CBR* é a criação da estrutura de um caso, ou seja, os atributos que serão utilizados como descrição do problema de aplicação e descrição da solução de um problema de aplicação. Para isso, deve-se instanciar a classe Case e adicionar, nas suas listas de classes CaseFeature, novos atributos. Cada atributo é uma nova instância da classe CaseFeature. Para isso, no qual é necessário informar ao construtor desta classe, via parâmetro, além do o identificador, o nome e o tipo do novo atributo. Na aplicação *CBR*

desenvolvida para o municiamento de baterias, foram criados treze atributos (alguns deles são descritos na Tabela 4.1) como descrição do problema e três atributos (descritos na Tabela 4.2) como descrição da solução. Por exemplo, a Figura 4.2 demonstra como instanciar a classe Case e adicionar atributos na descrição do problema, instanciando a classe CaseFeature;

Figura 4.2 – Trecho de código referente a criação de um caso no *framework* de *CBR*

```
cbrCase = new Case();
cbrCase.caseDescription.Add(new CaseFeature(0, "Tipo do alvo", typeof(string), "AlvoTipo1"));
cbrCase.caseDescription.Add(new CaseFeature(1, "Distancia do alvo", typeof(float), 42.123));
```

Fonte: Código do *framework* de *CBR*

Tabela 4.1 – Alguns atributos presentes na descrição do problema de municiamento de uma bateria de artilharia

Índice	**Nome do atributo**	**Tipo do atributo**
0	Tipo do Alvo	string
1	Distancia do Alvo	float
2	Area do Alvo	float
3	Altitude da bateria de artilharia	float
4	Porcentagem de certeza	float
5	Porcentagem de destruicao	float
...	...	...

Tabela 4.2 – Atributos presentes na descrição da solução utilizada no municiamento de uma bateria de artilharia

Índice	**Nome do atributo**	**Tipo do atributo**
0	Tipo de munição utilizada	string
1	Numero de munições utilizados	int
2	Numero de unidades de artilharia utilizadas	int

- **Passo 3 (Construção da base de casos):** para popular a base de casos, é necessário atribuir valores aos atributos de um caso e adicionar ele na base de casos. Isso é realizado individualmente para todos os casos da base de casos. Na aplicação *CBR* desenvolvida para o municiamento de baterias de artilharia, foi construído um gerador automático de casos onde foram gerados valores para atribuídos de casos. Para gerar esses valores, as regras descritas no Capítulo 4.2 foram seguidas. Por exemplo, na Figura 4.2 é adicionado o valor para um atributo do caso ao mesmo tempo que o caso é criado (último parâmetro do construtor da classe CaseFeature) e, na Figura 4.3, esse caso gerado é adicionado na base de casos;

Figura 4.3 – Trecho de código referente a adição de um caso na base de casos

```
cbrAPI.AddCase(cbrCase);
```

Fonte: Código do *framework* de *CBR*

- **Passo 4 (Desenvolvimento de medidas de similaridade):** para construir uma aplicação de *CBR*, é necessário desenvolver medidas de similaridade global e local ou utilizar as medidas já existentes no *framework* de *CBR* desenvolvida. Na aplicação *CBR* desenvolvida para o municiamento de baterias, foi necessário criar as medidas de similaridades locais descritas no Capítulo 4.3. Por exemplo, na Figura 4.4 é apresentada a criação da medida de similaridade local "CertaintySimilarity";

Figura 4.4 – Trecho de código referente a criação de uma medida de similaridade local

```
public class CertaintySimilarity : AbstractLocalSimilarity
{

    public CertaintySimilarity()...

    public override float GetSimilarity(ConsultParams consultParams, Case searchCase, Case retrieveCase)...
}
```

Fonte: Código do *framework* de *CBR*

- **Passo 5 (Criação de uma estrutura de consulta):** uma vez que todas as medidas de similaridade a serem utilizadas estejam disponíveis, ainda é necessário criar a estrutura de uma consulta para a aplicação *CBR* sendo desenvolvida. Essa estrutura é criada a partir da instanciação da classe ConsultStructure. Por padrão, a medida de similaridade global utilizada nas consultas é a Distância Euclidiana. Contudo, é possível utilizar outra medida global de similaridade instanciando essa nova medida de similaridade global no atributo globalSimilarity da classe ConsultStructure. Esta classe também apresenta uma lista de objetos da classe ConsultParams, os quais devem ser adicionados para ser possível realizar uma consulta. Neste caso, essa classe contém um conjunto de informações necessárias para computar uma similaridade local. Essas informações são passadas via parâmetro no momento da instanciação da classe ConsultParams. Isso permite apresentar a informação sobre a) quais índices dos atributos do caso que são utilizados para computar a similaridade, b) o peso que esses atributos têm na análise de similaridade global e c) a medida de similaridade local que é utilizada para processar os atributos indicados. Por exemplo, cada linha da Tabela 4.3 apresenta um exemplo de parâmetros que são informados na instanciação da classe ConsultParams. Por exemplo, a Figura 4.5 apresenta um exemplo de como realizar a instanciação da estrutura do caso pela classe ConsultStructure e como informar a medida de similaridade global nesta estrutura. Na Figura 4.6, os parâmetros de consulta são adicionados à estrutura da consulta.

Figura 4.5 – Trecho de código referente instanciação da estrutura da consulta e instanciando uma nova medida de similaridade global

```
ConsultStructure consultStructure = new ConsultStructure();
consultStructure.globalSimilarity = new EuclideanDistance(consultStructure);
```

Fonte: Código do *framework* de *CBR*

Figura 4.6 – Trecho de código referente a adição de parâmetros de consulta na estrutura da consulta

```
consultStructure.consultParams.Add(new ConsultParams(new List<int>() { 2, 0 }, 1f, new AreaSimilarity()));
consultStructure.consultParams.Add(new ConsultParams(new List<int>() { 3 }, 1f, new LinearFunction(0f, 3000f)));
```

Fonte: Código do *framework* de *CBR*

Tabela 4.3 – Exemplo de parâmetros para a classe ConsultParams

Índices dos atributos	**Peso do parâmetro**	**Medida de similaridade Local**
0	1	Equals
0, 2	1	AreaSimilarity
4	1	LinearFunction
5	1	DestructionSimilarity
11	1	NumUnitsSimilarity
12	1	Equals
...	...	...

- **Passo 6 (Construção de consultas):** com a base de casos populada e a estrutura de uma consulta concluída, é possível executar a consulta. Para isso, o método Retrieve da classe *CBR*API (já instanciada) é utilizado, passando como parâmetro o caso que é utilizado como consulta, além da estrutura da consulta que é utilizada para medir a similaridade entre dois casos. Por exemplo, na Figura 4.7, é apresentado como realizar uma consulta com o *framework* de *CBR* e. A Figura 4.8 mostra um exemplo de consulta e de um caso recuperado por meio dela;

Figura 4.7 – Trecho de código referente a uma consulta no *framework* de *CBR*

```
List<Result> results = cbrAPI.Retrieve(searchCase, consultStructure);
```

Fonte: Código do *framework* de *CBR*

- **Passo 7 (Utilização das respostas de uma consulta):** ao utilizar o método Retrieve, uma lista de resultados (Figura 4.7) organizados pela classe Result podem ser utilizados pela aplicação *CBR* desenvolvida. No caso da aplicação de municiamento de uma bateria de artilharia, a solução contida no caso mais similar recuperado da base de casos é utilizada

Figura 4.8 – Exemplo de consulta em um sistema de *CBR*

Consulta

Descrição do problema	
Atributo	**Valor**
Tipo do alvo	AlvoTipo5
Distancia do alvo	11.618
Area do Alvo	2.5
Numero de MunicaoTipo1 disponivel	88
Numero de unidades de artilharia disponiveis	4
…	…

Análise de similaridade

Caso recuperado

Descrição do problema	
Atributo	**Valor**
Tipo do alvo	AlvoTipo5
Distancia do alvo	13.182
Area do Alvo	2.5
Numero de MunicaoTipo1 disponivel	75
Numero de unidades de artilharia disponiveis	5
…	…
Solução proposta	
Atributo	**Valor**
Municao Utilizada	MunicaoTipo 1
Numero de municoes utilizadas	68
Numero de unidades de artilharia utilizadas	3

para automaticamente resolver o problema de municiamento da bateria utilizada como consulta. A Figura 4.9 mostra um exemplo de reutilização de uma solução passada em um novo problema (consulta);

4.2 REGRAS PARA A CRIAÇÃO DA BASE DE CASOS

Para resolver o problema de municiamento de baterias de artilharia, foi necessário criar uma base de casos de municiamento para o cumprimento de diferentes missões de tiro. No total, foram criados 1000 (mil) diferentes casos para essa aplicação de *CBR*. Assim como descrito anteriormente, cada caso possui 13 atributos para a descrição do problema (Tabela 4.1) e 3 atributos para a descrição da solução (Tabela 4.2). Um exemplo de caso presente no arquivo da base de casos pode ser visto na Figura 4.8. Para a criação desses casos, um gerador automático de casos para esse problema de aplicação foi construído. Baseado em um estudo sobre este problema de municiamento, e utilizando uma estratégia de raciocínio regressivo (dada uma solução para uma missão de tiro, quais informações deveriam estar presentes na descrição de um caso para que aquela solução pudesse acontecer), este gerador de casos obedece às seguintes regras para a criação de casos:

Figura 4.9 – Exemplo de reutilização de uma solução proposta após a recuperação

Consulta

Descrição do problema	
Atributo	**Valor**
Tipo do alvo	AlvoTipo5
Distancia do alvo	11.618
Area do Alvo	2.5
Numero de MunicaoTipo1 disponivel	88
Numero de unidades de artilharia disponiveis	4
…	…
Descrição da solução	
Atributo	**Valor**
Municao Utilizada	MunicaoTipo1
Numero de municoes utilizadas	68
Numero de unidades de artilharia utilizadas	3

- **Regra 1:** cada caso gerado deve ter um tipo de alvo a ser alvejado em uma missão de tiro. Existem vários tipos de alvos, nomeados como AlvoTipo1, AlvoTipo2, AlvoTipo3 e assim por diante;
- **Regra 2:** cada caso gerado deve ter entre 1 a 3 diferentes tipos de munição disponíveis para serem carregadas em uma bateria de artilharia. Esses tipos de munição são nomeados como MunicaoTipo1, MunicaoTipo2 e MunicaoTipo3;
- **Regra 3:** cada caso deve ter apenas um tipo de missão de tiro. Existem três tipos de missões, nomeadas como MissaoTipo1, MissaoTipo2 e MissaoTipo3. Os tipos de missões são gerados a partir do tipo de munição que será utilizada, por exemplo, em situações onde o único tipo de munição disponível para cumprir uma missão seja MunicãoTipo2, somente são gerados casos contendo as missões do tipo MissaoTipo1 e MissaoTipo2, nos demais casos de tipos de munição, podem ser gerados qualquer um dos três tipos de missões;
- **Regra 4:** a distância do alvo depende do tipo de missão de tiro e dos tipos munições disponíveis para serem carregadas na bateria de artilharia;
- **Regra 5:** a partir da distância do alvo, do tipo de missão de tiro e dos tipos de munição disponíveis, é calcula a altitude da bateria de artilharia;

- **Regra 6:** a área do alvo depende do tipo de missão de tiro e das munições disponíveis para serem utilizadas;
- **Regra 7:** o nível de certeza de um tiro é influenciado por informações sobre a realização de tiros de pontaria e a análise de condições meteorológicas, as quais são geradas aleatoriamente. A partir dessas informações, o nível de certeza de um tiro é reduzido por uma porcentagem. Quanto melhor forem as informações para apoiar um tiro, um caso deve ser baseado em uma redução da quantidade de munições utilizadas;
- **Regra 8:** a quantidade de munição que é utilizada no tiro depende do número de munições disponíveis para serem utilizadas, a área do alvo, o nível de certeza almejado para o tiro e a porcentagem de destruição almejado para o alvo;
- **Regra 9:** o número de unidades de artilharia utilizadas em uma missão de tiro depende da quantidade e o tipo de munição que é utilizada para cumprir a missão;
- **Regra 10:** o número e tipo de munições disponíveis depende do número e tipo de munições a serem utilizadas em uma missão. Por exemplo, se forem utilizadas 10 munições do tipo MunicãoTipo1 em uma missão, o caso gerado deve ter 10 ou mais munições deste tipo disponíveis;
- **Regra 11:** o número de unidades de artilharia disponíveis depende do número de unidades de artilharia a serem utilizadas para cumprir uma missão. Por exemplo, se 4 unidades de artilharia devem ser utilizadas para cumprir uma missão, o caso gerado deve ter 4 ou mais unidades de artilharia disponíveis.

Além das regras para criar um caso, o caso gerado automaticamente só pode ser adicionado na base de casos se apresentar 6 ou menos unidades de artilharia utilizadas.

4.3 MEDIDAS DE SIMILARIDADE LOCAIS DESENVOLVIDAS

No problema de aplicação de municiamento de uma bateria de artilharia foram criadas medidas de similaridades locais específicas para alguns atributos. Estas medidas são descritas pelos seguintes algoritmos:

- **AreaSimilarity (Algoritmo 5):** a ideia é que a diferença de área dos dois casos analisados não pode ser superior a 1 quilômetro quadrado, para evitar utilizar munição insuficiente ou desperdiçar munição ao alvejar um alvo. Assim, caso essa diferença for inferior a 1 quilômetro quadrado, a similaridade retornada é um, ou seja, a solução é possivelmente utilizável. Caso contrário, é retornado 0;

Algoritmo 5: AREASIMILARITY

Entrada: $consultParams, searchCase, retrieveCase$

Saída: Valor de similaridade entre 0 e 1

```
1  início
2      searchCaseArea ← GetArea(seachCase)
3      retrieveCaseArea ← GetArea(retrieveCase)
4      areasDifference ← retrieveCaseArea − searchCaseArea
5      se areasDifference > 1 então
6          retorna 0
7      fim
8      se areasDifference < −1 então
9          retorna 0
10     fim
11     retorna 1
12 fim
```

- **DestructionSimilarity (Algoritmo 6):** como o algoritmo anterior, o objetivo é não utilizar mais munição do que o necessário e, também, não utilizar munição insuficiente para alvejar o alvo. Assim, o Algoritmo 6 retornará 0, caso algum dos casos apresente uma porcentagem de destruição acima de 20% e o outro 20% ou menos. Nos demais casos, retornará 1;

Algoritmo 6: DESTRUCTIONSIMILARITY

Entrada: $consultParams, searchCase, retrieveCase$

Saída: Valor de similaridade entre 0 e 1

```
1  início
2      searchCaseDestruction ← GetDestruction(seachCase)
3      retrieveCaseDestruction ← GetDestruction(retrieveCase)
4      se searchCaseDestruction ≤ 20 & retrieveCaseDestruction > 20 então
5          retorna 0
6      fim
7      se retrieveCaseDestruction ≤ 20 & searchCaseDestruction > 20 então
8          retorna 0
9      fim
10     retorna 1
11 fim
```

- **CertaintySimilarity (Algoritmo 7):** a porcentagem de certeza é reduzida caso haja análise da condição climática ou tiro de teste (10% para cada) até um mínimo de 50%. Após estes descontos, tanto no atributo do caso de consulta como no caso retornado, é retor-

nada uma função linear aplicada estes valores. Assim, reduzindo o número de munição necessária;

Algoritmo 7: CERTAINTYSIMILARITY

Entrada: $consultParams, searchCase, retrieveCase$

Saída: Valor de similaridade entre 0 e 1

início

$searchCaseCertainty \leftarrow GetCertainty(seachCase)$

$retrieveCaseCertainty \leftarrow GetCertainty(retrieveCase)$

$searchCaseHasTestShot \leftarrow GetHasTestShot(seachCase)$

$retrieveCaseHasTestShot \leftarrow GetHasTestShot(retrieveCase)$

$searchCaseHasWeatherAnalysis \leftarrow GetHasWeatherAnalysis(seachCase)$

$retrieveCaseHasWeatherAnalysis \leftarrow GetHasWeatherAnalysis(retrieveCase)$

$searchCaseCertainty \leftarrow searchCaseCertainty - (searchCaseHasTestShot * 10) - (searchCaseHasWeatherAnalysis * 10)$

$retrieveCaseCertainty \leftarrow retrieveCaseCertainty - (retrieveCaseHasTestShot * 10) - (retrieveCaseHasWeatherAnalysis * 10)$

se $searchCaseCertainty < 50$ **então**

  $searchCaseCertainty \leftarrow 50$

fim

se $retrieveCaseCertainty < 50$ **então**

  $retrieveCaseCertainty \leftarrow 50$

fim

$similarity \leftarrow 1 - \frac{|\, searchCaseCertainty - retrieveCaseCertainty \,|}{99 - 50}$

retorna $similarity$

fim

- **NumUnitsSimilarity (Algoritmo 8):** o número de unidades de artilharia que deverão ser utilizadas deve ser menor ou igual ao número de disponíveis. Caso isso seja verdadeiro, é retornado 1. Caso contrário, é retornado 0.

Algoritmo 8: NUMUNITSSIMILARITY

Entrada: $consultParams, searchCase, retrieveCase$

Saída: Valor de similaridade entre 0 e 1

```
1 início
2     numAvailableUnits ← GetAvailableUnits(seachCase)
3     numUtilizedUnits ← GetUtilizedUnits(retrieveCase)
4     se numAvailableUnits < numUtilizedUnits então
5         retorna 0
6     senão
7         retorna 1
8     fim
9 fim
```

4.4 REGRAS PARA ANÁLISE DOS CASOS RETORNADOS PARA UMA CONSULTA DADA

Para resolver o problema de municiamento de baterias de artilharia, regras também são utilizadas para avaliar se o caso mais similar retornado para uma consulta pode resolver satisfatoriamente o problema dado ou não. Em geral, essas regras são utilizadas para avaliar a acurácia do processo de recuperação implementado na aplicação de municiamento. Entre essas regras, temos:

- **Regra 1:** se o tipo de missão do caso de consulta e do caso mais similar retornado da base de casos forem diferentes, a solução não pode ser aplicada ao caso usado como consulta. Diante deste fato, o sistema *CBR* não soluciona o problema dado (um erro na medida de acurácia do sistema deve ser contabilizado);

- **Regra 2:** se a diferença entre as áreas dos alvos do caso usado como consulta e do caso mais similar retornado da base de casos for maior que 1 quilômetro quadrado, a solução não pode ser aplicada ao caso usado como consulta. Nesta situação, um erro na medida de acurácia do sistema deve ser contabilizado;

- **Regra 3:** se o nível de destruição do alvo do caso usado como consulta for menor ou igual a 20% e o nível de destruição do caso mais similar retornado da base de casos for maior que 20% ou se o nível de destruição do caso mais similar retornado da base de casos for menor ou igual a 20% e o nível de destruição do caso usado como consulta for maior que 20%, a solução não pode ser aplicada ao caso usado como consulta. Nesta situação, novamente, um erro é contabilizado;

- **Regra 4:** se o número de munições utilizadas pelo caso mais similar retornado da base de casos for maior que o número de munições disponíveis no caso usado como consulta, a solução não pode ser aplicada ao caso usado como consulta. Nesta situação, novamente, um erro é contabilizado;

- **Regra 5:** se o número de unidades de artilharia utilizadas pelo caso mais similar retornado da base de casos for maior que o número de unidades de artilharia disponíveis no caso usado como consulta, a solução não pode ser aplicada ao caso usado como consulta. Nesta situação, novamente, um erro é contabilizado;

- **Regra 6:** o caso recuperado só é considerado válido se obter uma porcentagem de similaridade com o caso de consulta acima de 85%;

Nas demais situações, para fins de avaliação do mecanismo de recuperação implementado na aplicação de municiamento de uma bateria, o sistema *CBR* vai acertar o municiamento necessário para a bateria. Nesta situação, um acerto é contabilizado na medida de acurácia do sistema *CBR*.

4.5 RESULTADOS OBTIDOS NA APLICAÇÃO *CBR* PARA MUNICIAMENTO DE UMA BATERIA DE ARTILHARIA

Para analisar a acurácia da aplicação desenvolvida, foi utilizado o método *Leave one out and test* no formato do Algoritmo 9. Este método segue o seguinte ciclo: para cada caso da base de casos, remove o caso e o utiliza como consulta no passo de recuperação do ciclo de *CBR*. Caso a solução retornada pelo processo de recuperação (função Retrieve do Algoritmo 9) possa ser utilizada para solucionar o problema do caso utilizado como consulta (função Test do Algoritmo 9, baseada nas regras descritas na Seção 4.4), é adicionado 1 ao somatório de casos solucionados (variável sum do Algoritmo 9). Após isso, o caso utilizado como consulta é adicionado navamente a base de casos. Depois de todos os casos da base de casos serem utilizados como consulta, o somatório de casos solucionados é dividido pelo número de casos da base de casos. Assim, obtendo a acurácia da aplicação desenvolvida.

Utilizando o Algoritmo 9, foram criados dois tipos diferentes de avaliação para esta aplicação:

- Na primeira avaliação foram utilizadas as medidas de similaridades locais já existentes no *framework* de *CBR* desenvolvido (o Algoritmo 1 para os atributos numéricos e o Algoritmo 2 para os atributos de texto e booleanos). Nesta avaliação, foi obtido uma acurácia de 50.5%;

Algoritmo 9: ACURÁCIA

```
Entrada: caseBase
Saída: Valor de acurácia entre 0 e 1
1  início
2      para cada case ∈ caseBase faça
3          sum ← 0
4          Remove(case, caseBase)
5          retrieveCase ← Retrieve(case, caseBase)
6          se Teste(case, retrieveCase) então
7              sum++
8          fim
9          Adiciona(case, caseBasae)
10     fim
11     retorna sum/Tamanho(caseBase)
12 fim
```

- Na segunda avaliação foram utilizadas medidas de similaridades locais específicas para alguns atributos (descritas no Capítulo 4.3). Nesta avaliação, foi obtido uma acurácia de 66.3%.

A partir destas duas avaliações, concluímos como medidas de similaridade otimizadas, para o problema de aplicação, são extremamente importantes para obter uma acurácia melhor. Além disso, é possível melhorar ainda mais esta acurácia ajustando os pesos de cada parâmetro da consulta e, também, melhorando as medidas de similaridade locais desenvolvidas.

A acurácia deste problema de aplicação mostrou-se superior a acurácia dos estudos de caso de outros trabalhos relacionados (Capítulo 2.3). Por exemplo, os estudos de caso de Ontañón et al. (2010) e Sugandh, Ontañón e Ram (2008) obtiveram uma acurácia de 17.08% e 10% (sem utilizar adaptação), respectivamente.

4.6 DESAFIOS ENCONTRADOS NO DESENVOLVIMENTO DO ESTUDO DE CASO

Na construção da aplicação sobre o municiamento de baterias de artilharia utilizando o *framework* de CBR descrito neste trabalho, diversos desafios surgiram, como;

- **Informações para representar os casos na aplicação *CBR* alvo:** um sistema *CBR* necessita que os casos estejam bem representados, ou seja, que as informações relevantes para a representar as características do problema e solução sejam disponibilizadas por pessoas que sejam treinadas na solução destes problemas. Por exemplo, no domínio de simulação militar esse tipo de informação é bastante escasso (ou quase nulo), sendo necessário consultar manuais para extrair o máximo de informações possíveis. Contudo,

sem o devido treinamento na execução de tarefas de municiamento, a compreensão desses manuais foi bastante prejudicada. Em geral, a aplicação *CBR* desenvolvida para solucionar este problema de municiamento apenas utiliza informações bastante básicas sobre regras e doutrinas a serem utilizadas na solução de problemas de municiamento;

- **Relevância dos atributos representados nos casos**: sistemas *CBR* que utilizam pesos (nível de importância) iguais a 1 para todos os atributos indexados geralmente apresentam uma acurácia baixa. Na prática, existem atributos que não são tão importantes para a análise de similaridade e acabam influenciando negativamente a utilidade do resultado retornado. Devido a esse fato, é necessária uma ajustagem nesses pesos para que possa ocorrer um aumento de acurácia no sistema *CBR* resultante. Na aplicação *CBR* para municiamento de baterias, informações relevantes para possibilitar o ajuste desses pesos não estava disponível (usuários deste domínio de aplicação não estavam disponíveis para fomentar um melhor ajuste na relevância relativa desses atributos representados nos casos), como também um processo mais automatizado de ajuste de pesos não foi utilizado;

- **Construção da base de casos**: sistemas *CBR* aplicados a jogos de RTS, apresentados anteriormente, geralmente utilizam uma biblioteca que captura informações em replays de jogadores humanos. Com isso, um grande número de casos pode ser gerado e tratado para que possam compor a base de casos. Porém, bibliotecas que gravam as ações tomadas em um simulador tendem a não produzir um número de casos muito grande, onde problemas simulados têm naturezas distintas (em muitas aplicações de simulação, os mesmos problemas tendem a ser simulados várias vezes, o que tende a resultar nos mesmos casos).

5 CONCLUSÃO E TRABALHOS FUTUROS

Os trabalhos relacionados sobre *CBR* apresentam sistemas com técnicas bem fundamentadas, porém não são genéricos o suficiente para serem adaptados para outros problemas de aplicação a não ser a qual são designados. Já o *framework* de *CBR* mostrou-se capaz de ser genérico o suficiente para o desenvolvimento de novas aplicações por meio dela, porque permite estendê-la para a criação de novas funcionalidades específicas para um dado problema de aplicação.

Dentre outros aspectos, o *framework* de *CBR* permite criar um modelo de caso, construir uma base de casos a partir desse modelo e utilizar as funções de similaridade existentes para recuperar o caso mais similar. Além disso, por padrão, a Unity não apresenta técnicas de aprendizado de máquina. Assim, o *framework* de *CBR* se torna tanto uma novidade para a engine gráfica como um ótimo componente para a criação de sistemas inteligentes em aplicações desenvolvidas na Unity (comprovado pela aplicação de municiamento de baterias de artilharia).

Ainda existem uma série de limitações presentes no *framework* de *CBR* desenvolvido. Dentre elas temos a incapacidade de lidar com tipos de dados mais complexos (e.g. estruturas de dados), a inexistência de técnicas automatizadas para as etapas de reutilização, revisão e retenção do ciclo de *CBR* e o desempenho da consulta em relação ao crescimento da base de casos. Para isso, em futuras versões deste *framework*, técnicas utilizadas em sistemas de *CBR* presentes em jogos de RTS podem ser melhor exploradas para resolver estas limitações.

Entre trabalhos futuros, um problema de aplicação alvo a ser atacado é a revisão pós-ação. Nele, um sistema *CBR* pode ser capaz de utilizar experiências passadas de revisão pós-ação para dar apoio a decisão na análise das tarefas realizadas em simulações correntes.

REFERÊNCIAS BIBLIOGRÁFICAS

AAMODT, A.; PLAZA, E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. **AI Commun.**, v. 7, n. 1, p. 39–59, 1994.

BERGMANN, R. **Experience Management: Foundations, Development Methodology, and Internet-Based Applications**. [S.l.]: Springer, 2002. v. 2432. (Lecture Notes in Computer Science, v. 2432).

BERGMANN, R.; KOLODNER, J. L.; PLAZA, E. Representation in case-based reasoning. **Knowledge Eng. Review**, v. 20, n. 3, p. 209–213, 2005.

BORCK, H. et al. Case-based behavior recognition in beyond visual range air combat. In: **FLAIRS Conference**. [S.l.]: AAAI Press, 2015. p. 379–384.

CADENA, P.; GARRIDO, L. Fuzzy case-based reasoning for managing strategic and tactical reasoning in starcraft. In: **MICAI (1)**. [S.l.]: Springer, 2011. (Lecture Notes in Computer Science, v. 7094), p. 113–124.

CAYIRCI, E. Multi-resolution federations in support of operational and higher level combined/joint computer assisted exercises. In: **Winter Simulation Conference**. [S.l.]: WSC, 2009. p. 1787–1797.

ERIKSSON, J.; TORNES, D. Ø. **Learning to play Starcraft with Case-based Reasoning**. 2012. Dissertação (Mestrado) — Norwegian University of Science and Technology, 2012.

EVERTSZ, R. et al. Modeling rules of engagement in computer-generated forces. In: **Proceedings of the 16th Conference on Behavior Representation in Modeling and Simulation**. Orlando, FL: U. of Central Florida. 07-BRIMS-021. Orlando, FL: U. of Central Florida. 07-BRIMS-021, 2007. p. 123–134. Disponível em: <http://acs.ist.psu.edu/papers/evertszRRS07.pdf>.

FLETCHER, J. D. Education and training technology in the military. **Science**, v. 323, p. 72–75, 2009.

HEINZE, C. et al. Interchanging agents and humans in military simulation. In: **IAAI**. [S.l.]: AAAI, 2001. p. 27–34.

HODSON, D. D.; HILL, R. R. The art and science of live, virtual, and constructive simulation for test and analysi. **The Journal of Defense Modeling Simulation**, v. 11, n. 2, p. 77–89, 2013.

JONES, R. M. et al. Automated intelligent pilots for combat flight simulation. **AI Magazine**, v. 20, n. 1, p. 27–41, 1999.

LAIRD, J.; VANLENT, M. Human-level AI's killer application: Interactive computer games. **AI magazine**, v. 22, n. 2, p. 15, 2001.

LARA-CABRERA, R.; COTTA, C.; LEIVA, A. J. F. A review of computational intelligence in RTS games. In: **FOCI**. [S.l.]: IEEE, 2013. p. 114–121.

LOPES, R.; BIDARRA, R. Adaptivity challenges in games and simulations: A survey. **IEEE Trans. Comput. Intellig. and AI in Games**, v. 3, n. 2, p. 85–99, 2011.

MÁNTARAS, R. L. de; CUNNINGHAM, P.; PERNER, P. Emergent case-based reasoning applications. **Knowledge Eng. Review**, v. 20, n. 3, p. 325–328, 2005.

MÁNTARAS, R. L. de et al. Retrieval, reuse, revision and retention in case-based reasoning. **Knowledge Eng. Review**, v. 20, n. 3, p. 215–240, 2005.

MICHAEL, D. R.; CHEN, S. L. **Serious Games: Games That Educate, Train, and Inform**. [S.l.]: Muska & Lipman/Premier-Trade, 2005. ISBN 1592006221.

ONTAÑÓN, S. et al. On-line case-based planning. **Computational Intelligence**, v. 26, n. 1, p. 84–119, 2010.

RAM, A.; ONTAÑÓN, S.; MEHTA, M. Artificial intelligence for adaptive computer games. In: **FLAIRS Conference**. [S.l.]: AAAI Press, 2007. p. 22–29.

REECE, D. A.; MCCORMACK, J.; ZHANG, J. A case-based reasoning tool for composing behaviors for computer generated forces. In: **BRIMS**. [S.l.]: BRIMS, 2004. p. –.

SUGANDH, N.; ONTAÑÓN, S.; RAM, A. On-line case-based plan adaptation for real-time strategy games. In: **AAAI**. [S.l.]: AAAI Press, 2008. p. 702–707.

TAMBE, M. et al. Intelligent agents for interactive simulation environments. **AI Magazine**, v. 16, n. 1, p. 15–39, 1995.

TECHNOLOGIES, U. **Unity3D**. 2016. Disponível em: <https://unity3d.com/pt>.

THE JSON Data Interchange Format. 1st edition. ed. [S.l.], 2013. Disponível em: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>.

WATSON, I. **Applying Knowledge Management: Techniques for Building Corporate Memories**. [S.l.]: Morgan Kaufmann, 2003. (Engineering village).

APÊNDICE A – DOCUMENTAÇÃO DO *FRAMEWORK* DE *CBR*

Como apêndice deste trabalho, encontra-se a documentação do código do *framework* de *CBR* desenvolvido. Esta documentação foi gerada automaticamente pela ferramenta Doxygen na versão 1.8.12.

CBRAPI

Gerado por Doxygen 1.8.12

Sumário

Capítulo 1

Índice da hierarquia

1.1 Hierarquia de classes

Esta lista de heranças está organizada, dentro do possível, por ordem alfabética:

Capítulo 2

Índice dos componentes

2.1 Lista de componentes

Lista de classes, estruturas, uniões e interfaces com uma breve descrição:

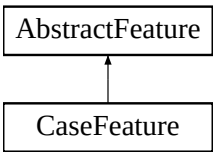
Capítulo 3

Documentação da classe

3.1 Referência à classe AbstractFeature

Classe abstrata que representa a estrutura básica de um atributo.

Diagrama de heranças da classe AbstractFeature



Membros públicos

- AbstractFeature (int id, string name, Type type)

  Construtor da classe AbstractFeature.

Atributos Públicos

- int id

  Identificador do atributo.
- string name

  Nome do atributo.
- string type

  Tipo de dado do atributo.

3.1.1 Descrição detalhada

Classe abstrata que representa a estrutura básica de um atributo.

3.1.2 Documentação dos Construtores & Destrutor

3.1.2.1 AbstractFeature()

```
AbstractFeature.AbstractFeature (
            int id,
            string name,
            Type type )
```

Construtor da classe AbstractFeature.

Parâmetros

id	Identificador do novo atributo.
name	Nome do novo atributo.
type	Tipo de dado do novo atributo.

3.1.3 Documentação dos dados membro

3.1.3.1 id

```
int AbstractFeature.id
```

Identificador do atributo.

3.1.3.2 name

```
string AbstractFeature.name
```

Nome do atributo.

3.1.3.3 type

```
string AbstractFeature.type
```

Tipo de dado do atributo.

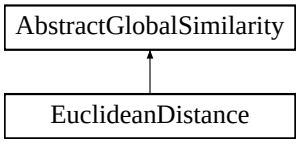
A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- AbstractClasses/AbstractFeature.cs

3.2 Referência à classe AbstractGlobalSimilarity

Classe abstrata que representa uma medida de similaridade global que será implementada na biblioteca CBR.

Diagrama de heranças da classe AbstractGlobalSimilarity



Membros públicos

- AbstractGlobalSimilarity (ConsultStructure consultStructure)

  Construtor da classe AbstractGlobalSimilarity.
- abstract float GetSimilarity (Case searchCase, Case retrieveCase)

  Método abstrato que deve ser implementado por todas as classes que extenderem a classe AbstractGlobalSimilarity.

Atributos Públicos

- ConsultStructure consultStructure

  Estrutura da consulta que será informada por cada classe que extender a classe AbstractGlobalSimilarity.

3.2.1 Descrição detalhada

Classe abstrata que representa uma medida de similaridade global que será implementada na biblioteca CBR.

3.2.2 Documentação dos Construtores & Destrutor

3.2.2.1 AbstractGlobalSimilarity()

```
AbstractGlobalSimilarity.AbstractGlobalSimilarity (
            ConsultStructure consultStructure )
```

Construtor da classe AbstractGlobalSimilarity.

Parâmetros

consultStructure	Estrutura da consulta.

3.2.3 Documentação dos métodos

3.2.3.1 GetSimilarity()

```
abstract float AbstractGlobalSimilarity.GetSimilarity (
            Case searchCase,
            Case retrieveCase )  [pure virtual]
```

Método abstrato que deve ser implementado por todas as classes que extenderem a classe AbstractGlobal↩ Similarity.

Parâmetros

searchCase	Caso utilizado como consulta.
retrieveCase	Caso recuperado da base de casos.

Retorna

Valor de similardade entre os casos c1 e c2.

Implementado em EuclideanDistance.

3.2.4 Documentação dos dados membro

3.2.4.1 consultStructure

```
ConsultStructure AbstractGlobalSimilarity.consultStructure
```

Estrutura da consulta que será informada por cada classe que extender a classe AbstractGlobalSimilarity.

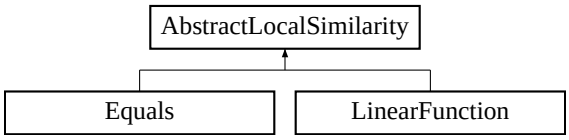
A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- AbstractClasses/AbstractGlobalSimilarity.cs

3.3 Referência à classe AbstractLocalSimilarity

Classe abstrata que representa uma medida de similaridade local que será implementada na biblioteca CBR.

Diagrama de heranças da classe AbstractLocalSimilarity



Membros públicos

- AbstractLocalSimilarity ()

  Construtor da classe AbstractLocalSimilarity.
- abstract float GetSimilarity (ConsultParams consultParams, Case searchCase, Case retrieveCase)

  Método abstrato que deve ser implementado por todas as classes que extenderem a classe AbstractLocalSimilarity.

3.3.1 Descrição detalhada

Classe abstrata que representa uma medida de similaridade local que será implementada na biblioteca CBR.

3.3.2 Documentação dos Construtores & Destrutor

3.3.2.1 AbstractLocalSimilarity()

```
AbstractLocalSimilarity.AbstractLocalSimilarity ( )
```

Construtor da classe AbstractLocalSimilarity.

3.3.3 Documentação dos métodos

3.3.3.1 GetSimilarity()

```
abstract float AbstractLocalSimilarity.GetSimilarity (
            ConsultParams consultParams,
            Case searchCase,
            Case retrieveCase )  [pure virtual]
```

Método abstrato que deve ser implementado por todas as classes que extenderem a classe AbstractLocalSimilarity.

Parâmetros

consultParams	Parâmetros da consulta.
searchCase	Caso utilizado como consulta.
retrieveCase	Caso recuperado da base de casos.

Retorna

Implementado em LinearFunction e Equals.

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- AbstractClasses/AbstractLocalSimilarity.cs

3.4 Referência à classe Case

Classe que representa um caso que compõem a base de casos. Composta por um identificador do caso e sua descrição do problema e solução.

Membros públicos

- Case ()

  Construtor da classe Case.

Atributos Públicos

- int id

  Identificador do caso.
- List< CaseFeature > caseDescription

  Descrição do problema a ser resolvido.
- List< CaseFeature > caseSolution

  Descrição da solução utilizada para resolver o problema.

3.4.1 Descrição detalhada

Classe que representa um caso que compõem a base de casos. Composta por um identificador do caso e sua descrição do problema e solução.

3.4.2 Documentação dos Construtores & Destrutor

3.4.2.1 Case()

```
Case.Case ( )
```

Construtor da classe Case.

3.4.3 Documentação dos dados membro

3.4.3.1 caseDescription

```
List<CaseFeature> Case.caseDescription
```

Descrição do problema a ser resolvido.

3.4.3.2 caseSolution

```
List<CaseFeature> Case.caseSolution
```

Descrição da solução utilizada para resolver o problema.

3.4.3.3 id

```
int Case.id
```

Identificador do caso.

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- Core/Case.cs

3.5 Referência à classe CaseBaseConnector

Classe resposável por fazer a conexão entre a biblioteca CBR e a base de casos.

Membros públicos

- CaseBaseConnector (string fileName)

  Construtor da classe CaseBaseConnector.
- void LoadCaseBase ()

  Método que carrega todos os casos da base de casos na memória RAM.
- void CreateCaseBase ()

  Método que cria a base de casos.
- Case LoadCase (int caseId)

  Método que carrega um caso da base de casos.
- void AddCase (Case newCase)

  Método que adiciona um novo caso na base de casos.
- void RemoveCase (int caseId)

  Método que remove um caso da base de casos.
- void EditCase (int caseId, Case newCase)

  Método que substitui um caso da base de casos por um novo caso.

Atributos Públicos

- List< Case > caseBase

  Lista de casos presentes na base de casos.

3.5.1 Descrição detalhada

Classe resposável por fazer a conexão entre a biblioteca CBR e a base de casos.

3.5.2 Documentação dos Construtores & Destrutor

3.5.2.1 CaseBaseConnector()

```
CaseBaseConnector.CaseBaseConnector (
            string fileName )
```

Construtor da classe CaseBaseConnector.

Parâmetros

fileName	Nome do arquivo que representa a base de casos.

3.5.3 Documentação dos métodos

3.5.3.1 AddCase()

```
void CaseBaseConnector.AddCase (
            Case newCase )
```

Método que adiciona um novo caso na base de casos.

Parâmetros

newCase	Novo caso a ser adicionado na base de casos.

3.5.3.2 CreateCaseBase()

```
void CaseBaseConnector.CreateCaseBase ( )
```

Método que cria a base de casos.

3.5.3.3 EditCase()

```
void CaseBaseConnector.EditCase (
            int caseId,
            Case newCase )
```

Método que substitui um caso da base de casos por um novo caso.

Parâmetros

caseId	Identificador do caso a ser alterado na base de casos.
newCase	Novo caso que alterará um caso presente na base de casos.

3.5.3.4 LoadCase()

```
Case CaseBaseConnector.LoadCase (
            int caseId )
```

Método que carrega um caso da base de casos.

Parâmetros

caseId	Identificador do caso a ser carregado.

Retorna

3.5.3.5 LoadCaseBase()

```
void CaseBaseConnector.LoadCaseBase ( )
```

Método que carrega todos os casos da base de casos na memória RAM.

3.5.3.6 RemoveCase()

```
void CaseBaseConnector.RemoveCase (
            int caseId )
```

Método que remove um caso da base de casos.

Parâmetros

caseId	Identificador do caso a ser removido da base de casos.

3.5.4 Documentação dos dados membro

3.5.4.1 caseBase

```
List<Case> CaseBaseConnector.caseBase
```

Lista de casos presentes na base de casos.

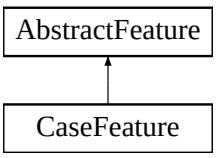
A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- Core/CaseBaseConnector.cs

3.6 Referência à classe CaseFeature

Classe que representa um atributo presente na descrição do problema ou solução de um caso.

Diagrama de heranças da classe CaseFeature



Membros públicos

- CaseFeature (int id, string name, Type type, object value)
  Construtor da classe CaseFeature.
- CaseFeature (int id, string name, Type type)
  Construtor da classe CaseFeature.

Atributos Públicos

- string value
  Valor do atributo.

3.6.1 Descrição detalhada

Classe que representa um atributo presente na descrição do problema ou solução de um caso.

3.6.2 Documentação dos Construtores & Destrutor

3.6.2.1 CaseFeature() [1/2]

```
CaseFeature.CaseFeature (
            int id,
            string name,
            Type type,
            object value )
```

Construtor da classe CaseFeature.

Parâmetros

id	Identificador do atributo.
name	Nome do atributo.
type	Tipo de dado do atributo.
value	Valor do atributo.

3.6.2.2 CaseFeature() [2/2]

```
CaseFeature.CaseFeature (
            int id,
            string name,
            Type type )
```

Construtor da classe CaseFeature.

Parâmetros

id	Identificador do atributo.
name	Nome do atributo.
type	Tipo de dado do atributo.

3.6.3 Documentação dos dados membro

3.6.3.1 value

```
string CaseFeature.value
```

Valor do atributo.

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- Core/CaseFeature.cs

3.7 Referência à classe CBRAPI

Classe que deve ser instanciada para ser possível utilizar a biblioteca CBR.

Membros públicos

- CBRAPI ()

  Construtor da classe CBRAPI.
- CBRAPI (string fileName)

  Construtor da classe CBRAPI.
- void AddCase (Case newCase)

  Método que passa um novo caso para a classe CaseBaseConnector adicionar na base de casos.
- void RemoveCase (int caseID)

  Método que informa a classe CaseBaseConnector qual caso deve ser removido.
- void EditCase (int caseID, Case newCase)

  Método que informa a classe CaseBaseConnector qual caso deve ser editado.
- List< Result > Retrieve (Case searchCase, ConsultStructure consultStructure)

  Realiza e etapa de recuperação do ciclo de CBR.

Atributos Públicos

- CaseBaseConnector caseBaseConnector

  Responsável por fazer a conexão entre a aplicação e a base de casos.

3.7.1 Descrição detalhada

Classe que deve ser instanciada para ser possível utilizar a biblioteca CBR.

3.7.2 Documentação dos Construtores & Destrutor

3.7.2.1 CBRAPI() [1/2]

```
CBRAPI.CBRAPI ( )
```

Construtor da classe CBRAPI.

3.7.2.2 CBRAPI() [2/2]

```
CBRAPI.CBRAPI (
            string fileName )
```

Construtor da classe CBRAPI.

Parâmetros

fileName	Nome do arquivo que represeta a base de casos.

3.7.3 Documentação dos métodos

3.7.3.1 AddCase()

```
void CBRAPI.AddCase (
            Case newCase )
```

Método que passa um novo caso para a classe CaseBaseConnector adicionar na base de casos.

Parâmetros

newCase	Novo caso.

3.7.3.2 EditCase()

```
void CBRAPI.EditCase (
            int caseID,
            Case newCase )
```

Método que informa a classe CaseBaseConnector qual caso deve ser editado.

Parâmetros

caseID	Identificador do caso e ser editado.
newCase	Novo caso que irá substituir um já existente.

3.7.3.3 RemoveCase()

```
void CBRAPI.RemoveCase (
            int caseID )
```

Método que informa a classe CaseBaseConnector qual caso deve ser removido.

Parâmetros

caseID	Identificador do caso a ser removido.

3.7.3.4 Retrieve()

```
List<Result> CBRAPI.Retrieve (
            Case searchCase,
            ConsultStructure consultStructure )
```

Realiza e etapa de recuperação do ciclo de CBR.

Parâmetros

searchCase	Caso de consulta.

Retorna

3.7.4 Documentação dos dados membro

3.7.4.1 caseBaseConnector

```
CaseBaseConnector CBRAPI.caseBaseConnector
```

Responsável por fazer a conexão entre a aplicação e a base de casos.

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- Core/CBRAPI.cs

3.8 Referência à classe ConsultParams

Classe que contém os parâmetros da consulta que serão utilizados para a análise de similaridade entre dois casos.

Membros públicos

- ConsultParams (List< int > indexes, float weight, AbstractLocalSimilarity localSimilarity)
  Construtor da classe ConsultParams.
- bool HasBlankFeature (Case searchCase)
  Verifica se algum atributo do caso de consulta está vazio

Atributos Públicos

- float weight
  Peso da consulta na análise de similaridade.
- AbstractLocalSimilarity localSimilarity
  Função de similaridade local que será utilizada na consulta.
- List< int > indexes
  Índice dos atributos utilizados na consulta.

3.8.1 Descrição detalhada

Classe que contém os parâmetros da consulta que serão utilizados para a análise de similaridade entre dois casos.

3.8.2 Documentação dos Construtores & Destrutor

3.8.2.1 ConsultParams()

```
ConsultParams.ConsultParams (
            List< int > indexes,
            float weight,
            AbstractLocalSimilarity localSimilarity )
```

Construtor da classe ConsultParams.

Parâmetros

indexes	Índice dos atributos utilizados na consulta.
weight	Peso da consulta na análise de similaridade.
localSimilarity	Função de similaridade local que será utilizada na consulta.

3.8.3 Documentação dos métodos

3.8.3.1 HasBlankFeature()

```
bool ConsultParams.HasBlankFeature (
            Case searchCase )
```

Verifica se algum atributo do caso de consulta está vazio

Parâmetros

searchCase	Caso de consulta

Retorna

O atributo está vazio ou não?

3.8.4 Documentação dos dados membro

3.8.4.1 indexes

```
List<int> ConsultParams.indexes
```

Índice dos atributos utilizados na consulta.

3.8.4.2 localSimilarity

```
AbstractLocalSimilarity ConsultParams.localSimilarity
```

Função de similaridade local que será utilizada na consulta.

3.8.4.3 weight

```
float ConsultParams.weight
```

Peso da consulta na análise de similaridade.

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- Core/ConsultParams.cs

3.9 Referência à classe ConsultStructure

Classe que contém a estrutura de consulta que será utilizada na análise de similaridade entre os casos.

Membros públicos

- ConsultStructure ()

  Construtor da classe ConsultStructure.
- ConsultStructure (AbstractGlobalSimilarity globalSimilarity)

  Construtor da classe ConsultStructure.

Atributos Públicos

- AbstractGlobalSimilarity globalSimilarity

  Função de similaridade global que será utilizada na análise de similaridade de dois casos.
- List< ConsultParams > consultParams

  Lista de todos os parâmetros de consulta utilizados na análise de similaridade.

3.9.1 Descrição detalhada

Classe que contém a estrutura de consulta que será utilizada na análise de similaridade entre os casos.

3.9.2 Documentação dos Construtores & Destrutor

3.9.2.1 ConsultStructure() [1/2]

```
ConsultStructure.ConsultStructure ( )
```

Construtor da classe ConsultStructure.

3.9.2.2 ConsultStructure() [2/2]

```
ConsultStructure.ConsultStructure (
            AbstractGlobalSimilarity globalSimilarity )
```

Construtor da classe ConsultStructure.

3.9.3 Documentação dos dados membro

3.9.3.1 consultParams

```
List<ConsultParams> ConsultStructure.consultParams
```

Lista de todos os parâmetros de consulta utilizados na análise de similaridade.

3.9.3.2 globalSimilarity

```
AbstractGlobalSimilarity ConsultStructure.globalSimilarity
```

Função de similaridade global que será utilizada na análise de similaridade de dois casos.

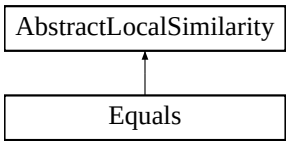
A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- Core/ConsultStructure.cs

3.10 Referência à classe Equals

Classe utilizada como função de similaridade local na qual compara se dois atributos são iguais.

Diagrama de heranças da classe Equals



Membros públicos

- Equals ()
  Construtor da classe Equals.
- override float GetSimilarity (ConsultParams consultParams, Case searchCase, Case retrieveCase)
  Método que retorna o valor de similaridade entre duas strings.

3.10.1 Descrição detalhada

Classe utilizada como função de similaridade local na qual compara se dois atributos são iguais.

3.10.2 Documentação dos Construtores & Destrutor

3.10.2.1 Equals()

```
Equals.Equals ( )
```

Construtor da classe Equals.

3.10.3 Documentação dos métodos

3.10.3.1 GetSimilarity()

```
override float Equals.GetSimilarity (
            ConsultParams consultParams,
            Case searchCase,
            Case retrieveCase )  [virtual]
```

Método que retorna o valor de similaridade entre duas strings.

Parâmetros

consultParams	Parâmetros da consulta.
searchCase	Caso utilizado como consulta.
retriveCase	Caso recuperado da base de casos.

Retorna

Valor de similaridade entre duas strings.

Implementa AbstractLocalSimilarity.

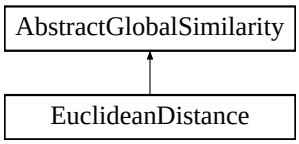
A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- SimilarityMeasures/LocalSimilarities/Equals.cs

3.11 Referência à classe EuclideanDistance

Classe que utiliza da função da Distância Euclidiana para calcular a similaridade global entre dois casos.

Diagrama de heranças da classe EuclideanDistance



Membros públicos

- EuclideanDistance (ConsultStructure consultStructure)

  Construtor da classe EuclideanDistance.
- override float GetSimilarity (Case searchCase, Case retrieveCase)

  Método que retorna o valor (entre 0 e 1) de similaridade entre dois casos utilizando a função da Distância Euclidiana.

Outros membros herdados

3.11.1 Descrição detalhada

Classe que utiliza da função da Distância Euclidiana para calcular a similaridade global entre dois casos.

3.11.2 Documentação dos Construtores & Destrutor

3.11.2.1 EuclideanDistance()

```
EuclideanDistance.EuclideanDistance (
            ConsultStructure consultStructure )
```

Construtor da classe EuclideanDistance.

Parâmetros

consultStructure	Estrutura de consulta que será utilizada na análise de similaridade.

3.11.3 Documentação dos métodos

3.11.3.1 GetSimilarity()

```
override float EuclideanDistance.GetSimilarity (
            Case searchCase,
            Case retrieveCase )  [virtual]
```

Método que retorna o valor (entre 0 e 1) de similaridade entre dois casos utilizando a função da Distância Euclidiana.

Parâmetros

searchCase	Caso utilizado como consulta.
retrieveCase	Casos recuperado da base de casos.

Retorna

Valor (entre 0 e 1) de similaridade entre dois casos

Implementa AbstractGlobalSimilarity.

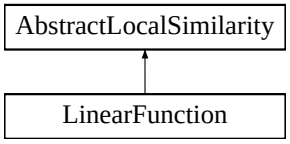
A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- SimilarityMeasures/GlobalSimilarities/EuclideanDistance.cs

3.12 Referência à classe LinearFunction

Classe que utilizar uma função linear para calcular a similaridade local entre dois atributos do caso.

Diagrama de heranças da classe LinearFunction



Membros públicos

- LinearFunction (float minVal, float maxVal)

  Construtor da classe LinearFunction.
- override float GetSimilarity (ConsultParams consultParams, Case searchCase, Case retriveCase)

  Método que retorna o valor de similaridade entre dois números.

3.12.1 Descrição detalhada

Classe que utilizar uma função linear para calcular a similaridade local entre dois atributos do caso.

3.12.2 Documentação dos Construtores & Destrutor

3.12.2.1 LinearFunction()

```
LinearFunction.LinearFunction (
            float minVal,
            float maxVal )
```

Construtor da classe LinearFunction.

Parâmetros

minVal	Valor mínimo do domínio do atributo.
maxVal	Valor máximo do domínio do atributo.

3.12.3 Documentação dos métodos

3.12.3.1 GetSimilarity()

```
override float LinearFunction.GetSimilarity (
            ConsultParams consultParams,
            Case searchCase,
            Case retriveCase )  [virtual]
```

Método que retorna o valor de similaridade entre dois números.

Parâmetros

consultParams	Parâmetros da consulta.
searchCase	Caso utilizado como consulta.
retriveCase	Caso recuperado da base de casos.

Retorna

Valor de similaridade entre dois números.

Implementa AbstractLocalSimilarity.

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- SimilarityMeasures/LocalSimilarities/LinearFunction.cs

3.13 Referência à classe Result

Classe que representa o resultado da consulta realizada na biblioteca CBR.

Membros públicos

- Result (int id, Case matchCase, float matchPercentage)

  Construtor da classe Result.

Atributos Públicos

- int id

  Identificador do resultado.
- Case matchCase

  Caso da base de casos que foi comparado.
- float matchPercentage

  Porcentagem de similaridade entre o caso de consulta e o caso da base de casos.

3.13.1 Descrição detalhada

Classe que representa o resultado da consulta realizada na biblioteca CBR.

3.13.2 Documentação dos Construtores & Destrutor

3.13.2.1 Result()

```
Result.Result (
            int id,
            Case matchCase,
            float matchPercentage )
```

Construtor da classe Result.

Parâmetros

id	Identificador do resultado.
matchCase	Caso da base de casos que foi comparado.
matchPercentage	Porcentagem de similaridade entre o caso de consulta e o caso da base de casos.

3.13.3 Documentação dos dados membro

3.13.3.1 id

```
int Result.id
```

Identificador do resultado.

3.13.3.2 matchCase

```
Case Result.matchCase
```

Caso da base de casos que foi comparado.

3.13.3.3 matchPercentage

```
float Result.matchPercentage
```

Porcentagem de similaridade entre o caso de consulta e o caso da base de casos.

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- Core/Result.cs

3.14 Referência à classe Search

Classe responsável por realizar o matching entre o caso de consulta e todos os casos da base de casos.

Membros públicos

- Search ()
  Construtor da classe Search.
- List< Result > DoSearch (Case searchCase, CaseBaseConnector caseBaseConnector, ConsultStructure consultStructure)
  Realiza o matching entre o caso de consulta e o caso da base de casos e retorna o resultado desse matching.
- List< Result > BubbleSort (List< Result > matchResults)
  Executa o algoritmo BubbleSort na lista de resultados.

3.14.1 Descrição detalhada

Classe responsável por realizar o matching entre o caso de consulta e todos os casos da base de casos.

3.14.2 Documentação dos Construtores & Destrutor

3.14.2.1 Search()

```
Search.Search ( )
```

Construtor da classe Search.

3.14.3 Documentação dos métodos

3.14.3.1 BubbleSort()

```
List<Result> Search.BubbleSort (
            List< Result > matchResults )
```

Executa o algoritmo BubbleSort na lista de resultados.

Parâmetros

matchResults	Lista de resultados.

Retorna

Lista de resultados ordenados.

3.14.3.2 DoSearch()

```
List<Result> Search.DoSearch (
            Case searchCase,
            CaseBaseConnector caseBaseConnector,
            ConsultStructure consultStructure )
```

Realiza o matching entre o caso de consulta e o caso da base de casos e retorna o resultado desse matching.

Parâmetros

searchCase	Caso de consulta.
caseBaseConnector	Conector da base de casos.
consultStructure	Estrutura de consulta que será utilizada.

Retorna

Lista de Result que contém o resultado de cada matching.

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- Core/Search.cs

Índice Remissivo