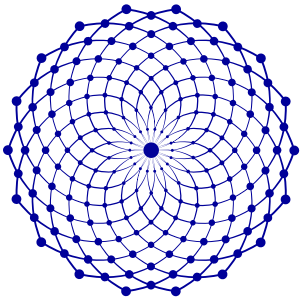


MODELAGEM COMPUTACIONAL DE CONHECIMENTO

Dissertação de Mestrado

MODELAGEM DE UM AMBIENTE PARA INICIANTES EM PROGRAMAÇÃO APOIADO POR UM ASSISTENTE INTELIGENTE

Cledson Calaça Cavalcante Gomes
cledsoncalaca@gmail.com

Orientadores:

Prof. Dr. Patrick Henrique da Silva Brito
Profa. Dra. Eliana Silva de Almeida

Maceió
2012

Cledson Calaça Cavalcante Gomes

MODELAGEM DE UM AMBIENTE PARA INICIANTES EM PROGRAMAÇÃO APOIADO POR UM ASSISTENTE INTELIGENTE

Dissertação de Mestrado apresentada ao Programa Multidisciplinar de Pós-Graduação em Modelagem Computacional de Conhecimento do Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para obtenção do título de Mestre em Modelagem Computacional.

Orientador: Prof. Dr. Patrick Henrique da Silva Brito

Coorientador: Profa. Dra. Eliana Silva de Almeida

Maceió
2012

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Fabiana Camargo dos Santos

G633m Gomes, Cledson Calaça Cavalcante.

Modelagem de um ambiente para iniciantes em programação apoiado por um assistente inteligente / Cledson Calaça Cavalcante Gomes. – 2012.

122 f. : il.

Orientador: Patrick Henrique da Silva Brito.
Coorientadora: Eliana Silva de Almeida.
Dissertação (Mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2012.

Bibliografia: f. 102–108.

1. Programação de computadores – Ensino. 2. Ferramenta de auxílio a aprendizagem. 3. Ensino-Aprendizagem. 4. Assistente inteligente. 5. Sistemas de informação – Educação à distância. I. Título.

CDU: 004.42:378

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Modelagem Computacional de Conhecimento
Avenida Lourival Melo Mota, Km 14, Bloco 09, Cidade Universitária
CEP 57.072-900 – Maceió – AL – Brasil
Telefone: (082) 3214-1364

Membros da Comissão Julgadora da Dissertação de Mestrado de Cledson Calaça Cavalcante Gomes, intitulada: "Modelagem de um Ambiente para Iniciantes em Programação Apoiado por um Assistente Inteligente", apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas em 13 de dezembro de 2012, às 08h30min, na sala de aula do Mestrado em Modelagem Computacional de Conhecimento.

COMISSÃO JULGADORA

Prof. Dr. Patrick Henrique da Silva Brito
UFAL – Instituto de Computação
Orientador

Profa. Dra. Eliana Silva de Almeida
UFAL – Instituto de Computação
Orientadora

Prof. Dr. Evandro de Barros Costa
UFAL – Instituto de Computação
Examinador

Prof. Dr. Jair Cavalcanti Leite
UFRN – Departamento de Informática e Matemática Aplicada
Examinador

Maceió, dezembro de 2012.

AGRADECIMENTOS

Agradeço primeiramente a Deus por me conceder a vida e por me dar forças para superar todas as dificuldades encontradas em meu caminho.

Agradeço aos meus pais Noélia e Clênio que foram, sem dúvida alguma, meu alicerce na vida, fornecendo todo o apoio possível e ajudando na formação do meu caráter, além de não medirem esforços para me proporcionar uma educação de qualidade.

Agradeço a Islânia, por me apoiar e se manter sempre ao meu lado, inclusive nos momentos de dificuldade, e por compreender os momentos ausentes dedicados à conclusão deste trabalho.

Agradeço a todos os professores, em especial aos meus orientadores Eliana Almeida e Patrick Brito que se propuseram a compartilhar seus conhecimentos comigo e me orientaram na elaboração deste trabalho (cobrando, corrigindo, criticando, elogiando), me "aturando", de certa forma, durante essa minha jornada na Pós-Graduação.

Agradeço a todas as outras pessoas (irmão, família, amigos, colegas de turma, pessoal do LaCCAN) que direta ou indiretamente me acompanharam durante mais esta etapa da minha vida.

Obrigado a todos!

Cledson Calaça Cavalcante Gomes

RESUMO

Neste trabalho é apresentada uma proposta de ambiente computacional denominado AIIP - Ambiente Inteligente para Iniciantes em Programação, cujo objetivo é auxiliar alunos e professores no processo de ensino/aprendizagem de programação de computadores. O AIIP utiliza a abordagem de resolução de problemas, cujos problemas armazenados em sua base de dados são propostos para serem resolvidos pelos alunos durante o momento de interação com o ambiente. O AIIP possui ainda um assistente inteligente para auxiliar os alunos, acompanhando suas ações, fornecendo dicas e *feedbacks* apropriados durante o processo de resolução dos problemas, além de identificar os erros cometidos durante o processo de desenvolvimento de suas soluções algorítmicas. Com relação aos professores, o assistente inteligente também auxilia ao avaliar as soluções algorítmicas dos alunos de forma automática, de acordo com métricas pré-estabelecidas, passíveis de flexibilização. Após a realização de um experimento e aplicação de um questionário, ambos relacionados ao uso do AIIP, com alunos do curso de Sistemas de Informação à Distância da Universidade Federal de Alagoas, foram encontrados resultados que apontam para a viabilidade da proposta apresentada como ferramenta de auxilio a alunos e professores em turmas iniciais de programação de computadores.

Palavras-chave: Programação. Auxílio. Ensino/Aprendizagem. Assistente Inteligente.

ABSTRACT

This work presents a proposal for a computing environment called AIIP - Intelligent Environment for Beginners in Programming, whose objective is to help students and teachers in the teaching/learning process of computer programming. AIIP uses the problem solving approach, whose problems stored in their database are proposed to be solved by the students during their interactions with the environment. AIIP also has an intelligent assistant to help students, monitoring their actions, providing hints and appropriate feedbacks during the solving problems process, and identifying the mistakes while they are developing their algorithmic solutions. About teachers, the intelligent assistant also helps evaluating students's algorithmic solutions automatically according to pre-established metrics, capable of flexibility. After conducting an experiment and application of a survey, both related to the use of AIIP, with Information Systems students of Federal University of Alagoas, the obtained results indicate the feasibility of the proposal as tool to help students and teachers in initial classes of computer programming.

Keywords: Programming. Assistance. Teaching/Learning. Intelligent Assistant.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

AIIP Ambiente Inteligente para Iniciantes em Programação

CAI *Computer Aided Instruction*

ERBASE Escola Regional Bahia Alagoas Sergipe

GQM *Goal Question Metric*

IA Inteligência Artificial

ICAI *Intelligent Computer Aided Instruction*

IDE *Integrated Development Environment*

ILA Interpretador de Linguagem Algorítmica

ITA *Intelligent Tutoring Assistant*

PBL *Problem Based Learning*

ROLEP *Research on Learning Programming*

SBC Sociedade Brasileira de Computação

SBIE Simpósio Brasileiro de Informática na Educação

STI Sistemas Tutores Inteligentes

WEI *Workshop* sobre Educação em Computação

SUMÁRIO

1 INTRODUÇÃO

Este trabalho apresenta o AIIP - Ambiente Inteligente para Iniciantes em Programação. O AIIP é um ambiente que possui o objetivo de auxiliar alunos e professores no processo de ensino/aprendizagem de programação através da abordagem de resolução de problemas, apoiado por um assistente inteligente, que auxilia os alunos durante a interação com o ambiente, fornecendo dicas e *feedbacks* apropriados e avaliando de forma automática suas soluções algorítmicas de acordo com métricas pré-estabelecidas.

É possível acreditar em um consenso dos especialistas da área da computação com relação à classificação da programação de computadores como uma das disciplinas mais importantes da área. Porém, para a maioria dos alunos iniciantes, o aprendizado e o desenvolvimento de técnicas em programação parece ser um processo complexo, exigente e cheio de dificuldades, podendo acarretar possíveis problemas, como por exemplo, a desistência de alunos da disciplina e, na pior das hipóteses, do próprio curso.

Neste sentido, a utilização de técnicas de IA na elaboração e no desenvolvimento de ambientes de ensino-aprendizagem, devido às potencialidades destes tipos de sistemas, com relação ao auxílio no aprendizado do aluno, vem se tornando a cada dia objeto de estudo de pesquisadores das áreas da computação e da educação.

Entretanto, apesar de muitas pesquisas durante as últimas quatro décadas, e de recentes experiências bem sucedidas envolvendo empresas como Google e Microsoft, as promessas projetadas desde o início das pesquisas sobre os STI - Sistemas Tutores Inteligentes por Carbonell (1970), considerado por muitos pesquisadores como o pai destes sistemas, ainda não foram possíveis de serem atingidas, uma vez que o desenvolvimento de um STI em que o aluno receba auxílio personalizado e adequado ainda está um pouco distante de ser alcançado em sua totalidade devido a algumas razões, como por exemplo, a complexidade de *software* e as limitações de *hardware*.

Diante das dificuldades relacionadas a *software* e *hardware*, algumas alternativas foram e vem sendo propostas nos últimos anos, como, por exemplo, os ITA - *Intelligent Teaching Assistant*, ou Assistentes Inteligentes de Ensino. Os ITA são sistemas que possuem o objetivo de auxiliar, de forma inteligente, não somente os alunos, mas também os professores, além de disponibilizar auxílio e assistência individualizada aos alunos, facilitando também a disponibilização de materiais e exercícios personalizados. (Yacef, 2002)

Enquanto os STI possuem modelos mais complexos e estratégias pedagógicas mais invasivas, apresentando um maior número de interrupções do tutor na interação como aluno, os os ITA realizam tarefas mais simples, com um modelo de aluno simplificado e estratégias pedagógicas restritas baseadas no monitoramento do aluno e nos resultados obtidos ao longo da interação, com poucas interrupções.

Com relação às dificuldades relacionadas não só ao desenvolvimento destes sistemas, mas

também ao processo de ensino/aprendizagem de programação como um todo, o que realmente observa-se nos últimos anos é uma discussão acerca da utilização de novas metodologias de ensino com objetivo de melhorar este processo. Esta discussão tem sido foco de pesquisas no Brasil e no mundo há alguns anos. Diversas ferramentas, ambientes e metodologias são propostas com o objetivo de amenizar tais dificuldades e tornar o processo de ensino/aprendizagem de programação menos complexo.

O interesse da comunidade cientifica da área, em âmbito nacional, pode ser comprovado no trabalho de Júnior & Rapkiewicz (2006), onde verifica-se que pelo menos um artigo por ano sobre ensino/aprendizagem de programação é publicado tanto no SBIE - Simpósio Brasileiro de Informática na Educação, como no WEI - *Workshop* sobre Educação em Computação, eventos ligados à SBC - Sociedade Brasileira de Computação. Neste trabalho, os autores analisaram vários artigos e concluíram a existência de três vertentes na busca por soluções para as dificuldades vivenciadas por professores e alunos neste processo: as metodologias de ensino, as ferramentas e ambientes computacionais e a integração de ambas. Segundo os autores, apesar de concordarem que a abordagem que une ferramentas e metodologias apresenta melhores resultados, o que se observa na maioria dos trabalhos é uma tendência em tratá-las separadamente.

Sendo assim, este trabalho possui como objetivo geral a modelagem de um ambiente de ensino/aprendizagem de programação que utiliza a abordagem de resolução de problemas, apoiado por um assistente inteligente capaz de fornecer dicas e feedbacks e avaliar as soluções algorítmicas dos alunos de forma automática. Os objetivos específicos decorrentes do objetivo principal são: auxiliar alunos e professores no processo de ensino/aprendizagem de programação, prover boa usabilidade para o ambiente e prover flexibilidade no gerenciamento dos recursos do mesmo.

Expostos os objetivos, este trabalho propõe responder o seguinte problema de pesquisa: *a utilização de um ambiente de ensino/aprendizagem de programação, apoiado por um assistente inteligente, utilizando uma abordagem de resolução de problemas, pode auxiliar os alunos em seu aprendizado e os professores em suas tarefas de ensino?*

Para chegar às conclusões com relação ao problema proposto, este trabalho foi estruturado em 7 capítulos, incluindo esta introdução. No Capítulo 2 é apresentada uma breve discussão sobre o processo de ensino/aprendizagem de programação, considerando aspectos relacionados às dificuldades encontradas por alunos e professores, aspectos relacionados à escolha de uma linguagem para o ensino, além da apresentação de algumas metodologias de aprendizagem de programação, com foco nos assistentes inteligentes de ensino como alternativa à construção de um STI, apresentando sua arquitetura típica e algumas vantagens na utilização desta abordagem.

No Capítulo 3 são apresentados alguns trabalhos correlatos (ambientes e ferramentas) destinados ao ensino de programação, bem como algumas de suas características e abordagens metodológicas utilizadas para o ensino, contextualizando estes trabalhos com a proposta

apresentada nesta dissertação.

No Capítulo 4 o AIIP é apresentado. Suas características e arquitetura são descritas, juntamente com o detalhamento de seus componentes.

No Capítulo 5 o assistente inteligente do AIIP é apresentado, sendo possível entender como o mesmo pode auxiliar alunos e professores no processo de ensino/aprendizagem de programação através da apresentação de dicas e *feedbacks* apropriados e da avaliação automática, de acordo com métricas pré-estabelecidas, de suas soluções algorítmicas em reposta aos problemas armazenados no ambiente.

No Capítulo 6 é apresentado o experimento realizado para a avaliação do AIIP. O experimento foi dividido em três etapas: Planejamento, Teste Prático e Aplicação do Questionário e Análise dos Dados, e foi realizado com alunos do Curso de Sistemas de Informação à Distância da Universidade Federal de Alagoas e que já cursaram a disciplina de Algoritmos e Estrutura de Dados I.

Por fim, no Capítulo 7 são apresentadas as considerações finais, as principais contribuições deste trabalho e alguns direcionamentos com relação ao desenvolvimento de trabalhos futuros.

2 ASPECTOS RELACIONADOS AO ENSINO E APRENDIZAGEM DE PROGRAMAÇÃO

É possível afirmar que com o passar do tempo a programação de computadores deixou de ser considerada uma simples "arte", "*hobby*" ou "brincadeira" por parte de seus entusiastas, transformando-se em objeto de pesquisa e estudo em grandes centros e universidades em todo o mundo, envolvendo um grande conjunto de princípios, regras, técnicas e formalismos que visam à produção de programas bem estruturados e confiáveis.

O estudo da programação de computadores não se restringe apenas ao estudo das linguagens de programação. As linguagens de programação nada mais são do que ferramentas de concretização, que representam o resultado da aplicação de uma série de conhecimentos e transformam a especificação da solução de um problema em um programa de computador que efetivamente resolve o problema.

Existem várias formas de se representar um programa, ou um algoritmo, independente da linguagem a ser utilizada para representação. Fluxogramas e pseudocódigos são algumas destas alternativas. Nos fluxogramas os símbolos geométricos representam as estruturas de um programa, onde os símbolos são conectados por arestas dirigidas que fornecem uma sequência de execução do programa. Já o pseudocódigo permite representar um algoritmo de forma genérica, de modo mais informal, podendo ser interpretado por qualquer pessoa, sem a necessidade de se conhecer a sintaxe de nenhuma linguagem específica de programação.

Desta forma, de acordo com as diretrizes curriculares dos cursos da área de computação e informática do Ministério da Educação (1999), pode-se entender que a programação de computadores

> "é uma atividade voltada à solução de problemas. Nesse sentido, ela está relacionada com uma variada gama de outras atividades como especificação, projeto, validação, modelagem e estruturação de programas e dados, utilizando-se das linguagens de programação propriamente ditas, como ferramentas."

Autores como Schneider (1978), Deek et al. (1998), Raabe (2005) e Pears et al. (2007) também defendem a linha de pensamento que considera a programação como a aplicação de habilidades na resolução de problemas. Pears et al. (2007) afirmam que um aspecto chave na resolução de problemas é a capacidade de resolver duas classes de problemas: problemas similares a problemas já resolvidos e novos problemas. Deek et al. (1998) descrevem um cenário referente ao primeiro ano de um curso de programação baseado em um modelo de resolução de problemas, onde os recursos de linguagem são introduzidos apenas no contexto das soluções dos alunos aos problemas específicos.

Assim, a programação deve ser considerada muito mais do que a escrita de um conjunto de linhas de código em uma dada linguagem, e sim, uma atividade voltada à resolução de problemas, utilizando alguma linguagem que seja interpretável e que faça sentido, onde um conjunto de habilidades e conhecimentos é necessário para se chegar à solução dos problemas propostos.

2.1 Dificuldades no Ensino de Programação

A disciplina de programação é, sem dúvidas, uma das disciplinas chaves na formação do profissional da computação. No entanto, autores como Rocha (1991) e Winslow (1996) afirmam que aprender a programar é um processo naturalmente difícil. Para Winslow (1996), os programadores iniciantes sofrem com uma ampla variedade de dificuldades, onde os cursos de programação são geralmente aqueles que têm as maiores taxas de abandono. Além disso, segundo o mesmo autor, levam-se cerca de 10 anos para se transformar um programador iniciante em um programador especialista, ou seja, muito mais do que a média de 4 a 5 anos de duração de um curso de graduação na área da computação.

Já para Giraffa et al. (2003), são inúmeras as causas para os elevados índices de reprovação dos alunos nas disciplinas de algoritmos e programação, como por exemplo: hábitos de estudo centrados em memorização, falta de pré-requisitos em conteúdos correlatos, principalmente nos conteúdos de matemática, sequência didática desmotivadora, dificuldades na compreensão do enunciado dos problemas, forte carga de conceitos abstratos, cultura instrucionista e prática pedagógica desmotivadora.

Estas dificuldades podem ser percebidas não só pelos índices de reprovação na disciplina, mas também pelas dificuldades encontradas pelos alunos em outras disciplinas que exigem como pré-requisito a programação.

Desta forma, algumas perguntas nos levam a refletir sobre este processo, como por exemplo: Porque programação é difícil de aprender? Quais os aspectos cognitivos exigidos neste processo? Existe alguma estratégia de ensino bem-sucedida? O que pode ser feito para auxiliar o aprendizado de alunos iniciantes em programação?

O entendimento dos motivos de tanta dificuldade existente neste processo é explorado de diversas formas na literatura no mundo todo já há alguns anos.

De acordo com Rocha (1991), Robins et al. (2003) e Ala-Mutka (2004), alguns conceitos de programação deveriam receber uma atenção maior por serem considerados problemáticos para o entendimento dos alunos iniciantes. Alguns dos conceitos identificados por esses autores são: iteração, condição, recursão, passagem de parâmetros e matrizes.

Em seu estudo sobre erros em programas simples desenvolvidos na linguagem Pascal, Spohrer et al. (1985) também constataram que os erros associados com iterações e condições foram muito mais comuns do que aqueles associados com entrada e saída, inicialização,

atualização, sintaxe/estrutura de blocos e planejamento em geral.

Levantados alguns dos motivos técnicos, Jenkins (2002) acrescenta que os fatores motivacionais também devem ser levados em conta para o entendimento das dificuldades encontradas neste processo. Os alunos de computação podem ter um verdadeiro interesse no assunto (motivação intrínseca), outros podem ver o curso como um meio para uma carreira lucrativa (motivação extrínseca), enquanto outros podem estar simplesmente tentando agradar os pais ou a família (motivação social).

A heterogeneidade dos alunos em uma turma também é um fator que pode gerar dificuldade no processo de ensino/aprendizagem de programação. Ásrún Matthíasdóttir (2006) afirma que os alunos possuem diferentes níveis de maturidade, o que leva a diferentes hábitos de aprendizagem e diferentes motivações para estudar. Alguns alunos possuem um conhecimento interno nato e, consequentemente, uma facilidade natural para a elaboração de soluções algorítmicas para os problemas, em contrapartida, há aqueles que não possuem esta facilidade, e assim, o professor tem de se adaptar a cada aluno.

No trabalho apresentado por Winslow (1996) é realizada uma comparação entre alunos iniciantes em um curso de programação com programadores profissionais, onde o autor conclui que os alunos iniciantes são limitados quanto à utilização de conhecimento, não utilizam modelos mentais adequados, falham ao aplicar os conhecimentos, utilizam estratégias gerais ao invés de estratégias específicas na resolução de problemas e utilizam uma abordagem de programação linha por linha, ao invés de analisar os blocos e estruturas.

Já para Robins et al. (2003), em contraste com os especialistas, os iniciantes passam muito pouco tempo planejando e testando seus códigos, em compensação, os especialistas são mais rápidos, mais precisos, e capazes de recorrer a uma maior variedade de exemplos, estratégias e fontes de conhecimento.

A heterogeneidade dos cursos que incluem a programação como disciplina também é um fator que merece destaque. É difícil manter o mesmo padrão de ensino em cursos como engenharia da computação, ciência da computação, sistemas de informação, tecnologia da informação, análise de sistemas, ou seja, cursos da mesma área mas que ao mesmo tempo possuem perfis de alunos tão diferentes.

Porém, assim como para os alunos, existem também as dificuldades atreladas aos professores no processo de ensino de programação. Nobre & Menezes (2002) apresentam algumas das dificuldades vivenciadas pelo professor neste processo, são elas: I) reconhecer as habilidades inatas de seus alunos; II) apresentar técnicas para soluções de problemas; III) trabalhar a capacidade de abstração do aluno, tanto na busca das possíveis soluções como na escolha da estrutura de dados a ser utilizada; e IV) promover a cooperação e a colaboração entre os alunos.

De certa forma, estas dificuldades podem ser explicadas pelo fato dos professores terem que acompanhar um número grande de alunos em uma mesma turma. Outro problema comum que pode justificar as dificuldades encontradas pelos professores neste processo é a

falta de comunicação entre os docentes de outras disciplinas em que a programação esteja relacionada, visando uma troca de experiências e ideias, sejam estas disciplinas do mesmo semestre letivo ou de semestres posteriores.

Mesmo reconhecendo as dificuldades existentes no processo de ensino/aprendizagem de programação, acredita-se que o professor possui um papel fundamental neste processo, onde o mesmo deve ser o responsável por proporcionar novas situações e atividades de aprendizagem aos alunos, provocando assim, o desenvolvimento da capacidade de aprendizagem destes.

2.2 Escolha da Linguagem de Programação para o Ensino

Segundo Pears et al. (2007), a escolha da linguagem utilizada para o ensino geralmente é feita localmente, com base em alguns fatores, tais como: preferência do corpo docente, relevância para a indústria, aspectos técnicos da linguagem e disponibilidade de ferramentas e materiais úteis.

Atualmente é possível perceber que linguagens como C, C++ e Java são as linguagens de programação mais utilizadas, tanto no campo mercado/indústria como no campo da educação. Esta percepção é confirmada no sítio da comunidade de programação Tiobe[1], que apresenta mensalmente, através de motores de busca, uma lista atualizada da popularidade das diversas linguagens de programação utilizadas no mundo.

Apesar da popularidade de linguagens como Java, C e C++, tem havido muita discussão sobre a utilização destas linguagens no campo da educação, especialmente quando se trata da introdução à programação. De acordo com Pears et al. (2007), estas linguagens não foram projetadas especificamente para fins educacionais, em contraste com outras que foram criadas com esse objetivo específico, como por exemplo: Logo, Eiffel e Pascal.

Mesmo com relação às linguagens projetadas especificamente para o ensino, como por exemplo: Logo, Eiffel e Pascal, ainda existem algumas razões que nos levam a pensar sobre a inadequação da utilização das mesmas em um curso introdutório. Estas linguagens possuem um grande número de detalhes sintáticos e semânticos necessários para implementar suas funcionalidades, obrigando o aluno a se concentrar nestes detalhes ao invés de se concentrar no desenvolvimento da solução. Além disso, estas linguagens muitas vezes não permitem uma total abstração de aspectos da máquina, além de poder atrelar o aluno ao paradigma de programação a qual a linguagem utiliza.

Para Schneider (1978), a linguagem de programação utilizada no campo educacional deve ser baseada em dois aspectos críticos e aparentemente opostos: riqueza e simplicidade. Rica em prover as funcionalidades necessárias para a introdução dos conceitos fundamentais de programação, mas simples o suficiente para ser apresentada e compreendida em um semestre

[1] `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`

letivo.

Como pode-se perceber, a linguagem utilizada para o ensino de programação, independente do seu propósito de criação, possui um impacto importante no processo de ensino/aprendizagem. Neste sentido, sua escolha deve ser feita de forma sensata, levando em consideração, principalmente, o público-alvo a ser atendido e os recursos disponíveis para que a absorção do aprendizado seja a maior possível.

2.3 Algumas Estratégias de Ensino de Programação

Cada aluno possui sua origem, experiências e habilidades diferentes, e isto justifica, em parte, o fato de alunos de uma mesma classe, submetidos às mesmas condições de ensino, apresentarem resultados distintos. Desta forma, o processo de ensino/aprendizagem de programação deve ser trabalhado com cuidado pelos professores. Esses cuidados relacionam-se a diversos fatores, que vão desde o entendimento do problema abordado, do paradigma de programação selecionado para o ensino, passando pela linguagem escolhida e chegando às metodologias de ensino utilizadas.

Os estudos encontrados na literatura apresentam diferentes estratégias e abordagens na tentativa de melhorar o processo de ensino de programação. Apesar da existência de várias estratégias e abordagens, são apresentadas nesta seção apenas aquelas que possuem similaridade com a solução proposta neste trabalho.

2.3.1 Programação em Par

Na programação em par dois alunos trabalham em colaboração na construção do mesmo programa em um mesmo computador. Pode ser considerada uma forma de aprendizagem colaborativa, uma vez que os dois alunos envolvidos estão trabalhando em conjunto para alcançar o mesmo objetivo comum. Esta metodologia é abordada nos trabalhos de (Cockburn & Williams, 2001), (McDowell et al., 2006), (Han et al., 2006) e (Han et al., 2010).

A metodologia da programação em par é basicamente a seguinte: um dos alunos desempenha o papel de "desenvolvedor" do programa, enquanto o outro desempenha o papel de "observador", planejando as estratégias de resolução do problema. Para Han et al. (2010), este processo melhora a qualidade do código e tem um efeito positivo sobre a retenção e transferência de conhecimento por parte dos alunos através interação, da alternância dos papeis entre os alunos e da troca de opiniões diferentes sobre as estratégias de resolução dos problemas e outras estratégias.

Esta abordagem possivelmente poderia ajudar em uma situação de interação entre os tipos de alunos iniciantes identificados por Perkins et al. (1986). Para este autor existem 2 categorias de alunos iniciantes em programação, os "*stoppers*" e os "*movers*". Quando confrontados com um problema em que não conseguem prosseguir, os "*stoppers*" simplesmente param

de pensar e parecem abandonar as tentativas de solucionar o problema por conta própria. Em compensação, os "*movers*" sempre continuam tentando, experimentando e modificando o seu código, podendo usar *feedbacks* sobre seus erros cometidos de forma eficaz. Sendo assim, a junção destes tipos de alunos em pares poderia ajudar aos alunos "*stoppers*" a desenvolverem seu raciocínio, assim como fazem os "*movers*".

Apesar dos impactos educacionais vantajosos da programação em par, existem algumas dificuldades relacionadas ao uso dessa metodologia. Para Han et al. (2010), se um dos alunos que trabalham em par no mesmo computador tiver um horário radicalmente diferente do outro, será difícil para colocar em prática esta metodologia após as aulas. Além disso, a capacidade de programação de um dos pares pode afetar a metodologia, sendo uma tarefa difícil igualar os alunos em pares.

2.3.2 Aprendizagem Baseada em Problemas

A aprendizagem baseada em problemas, do inglês *Problem Based Learning* – PBL, é uma estratégia pedagógica de ensino em que os alunos trabalham com o objetivo de resolver um problema do mundo real, deixando de ser aprendizes passivos e passando a ser os principais responsáveis pelo seu aprendizado. (Hmelo-Silver, 2004)

De acordo com Hmelo-Silver (2004), na abordagem PBL os alunos devem trabalhar em pequenos grupos colaborativos e aprender o que precisam para resolver um problema. O professor deve agir como um facilitador para orientar a aprendizagem dos alunos através de um ciclo de aprendizagem.

Desta forma, a abordagem PBL vem sendo utilizada em diversas áreas de conhecimento pelo papel que proporciona aos alunos, à medida que estimula uma atitude ativa e não meramente informativa em busca do conhecimento, como observa-se nas práticas pedagógicas tradicionais.

No entanto, de acordo com Sousa (2011), a PBL aplicada nas áreas da educação em computação no Brasil é utilizada de forma bastante modesta. Geralmente, a abordagem PBL é introduzida parcialmente em algumas disciplinas da grade curricular do curso.

Porém, a PBL como estratégia de ensino na área da computação é foco de muitas pesquisas em outros países. O grupo de pesquisa ROLEP - *Research on Learning Programming* [1] da Universidade de Tecnologia de Helsinki na Finlândia, por exemplo, tem como foco principal a pesquisa sobre as experiências da aplicação da PBL em um curso introdutório de programação. De acordo com o grupo, a PBL é a principal estratégia de ensino e aprendizagem em um dos cursos introdutórios de programação da Universidade de Tecnologia de Helsinki, observando-se que a aprendizagem dos alunos usando a estratégia PBL vem ocorrendo com bons resultados.

[1] `http://www.cs.hut.fi/Research/COMPSER/ROLEP/ROLEP_paasivu.html`

Encorajado pelos resultados destas experiências, o grupo lançou uma variação da abordagem PBL, disponível no trabalho de Kinnunen & Malmi (2002), onde o papel dos tutores é reduzido radicalmente, e consequentemente a quantidade de recursos também. Esta variação foi criada para investigar como os alunos poderiam lidar com recursos de tutoria reduzidos, obtendo ainda benefícios da abordagem PBL.

Outro exemplo é o que foi aplicado no Departamento de Ciência da Computação da Universidade Nacional da Irlanda, que decidiu implementar a abordagem PBL em um módulo da disciplina de programação de computadores no primeiro ano do curso de Ciência da Computação. Embora não tenha sido notada nenhuma mudança drástica com relação ao aprendizado, Kelly et al. (2004) afirmam que os alunos aprenderam a trabalhar em grupo, respeitar compromissos, além de ouvir e perceber as diversas formas de solucionar um problema. Os primeiros resultados da introdução da abordagem PBL na disciplina foram positivos, levando o Departamento a manifestar o interesse em continuar utilizando esta abordagem.

2.3.3 Fornecimento de Dicas e *Feedbacks* em Tempo Real

Diversos estudos, como os de Aleven & Koedinger (2000), Vanlehn (2006) e Muñoz-Merino & Kloos (2009), têm demonstrado que o fornecimento de dicas durante a resolução de problemas tem sido uma estratégia bem sucedida no processo de aprendizagem. Vários sistemas implementam funções específicas relacionadas a este aspecto. A ideia comum no âmbito destes sistemas é de apresentar algum tipo de ajuda ao aluno, com relação a algum tema, não necessariamente fornecendo-o a solução final, mas imergindo-o em um processo de descoberta da resposta correta.

De acordo com Vanlehn (2006), a questão do fornecimento de dicas geralmente é decidida por uma política, como por exemplo, só fornecer dicas se o aluno solicitar explicitamente ou fornecer dicas ao detectar que o aluno está tendo dificuldades. Estas dicas geralmente são fornecidas em uma sequência, partindo de uma dica mais básica até uma dica mais específica. Independentemente de como as dicas sejam organizadas no sistema, fornecê-las de forma correta pode aumentar significativamente a aprendizagem dos alunos. (Vanlehn, 2006)

No entanto, ao se utilizar a política de fornecer dicas somente após a solicitação dos alunos, Aleven & Koedinger (2000) identificaram em seu trabalho dois problemas bastante interessantes: o "abuso de ajuda" e a "recusa de ajuda". No "abuso de ajuda", os alunos solicitam ajuda na forma de dicas mesmo quando não precisam, solicitando estas dicas repetidamente e em um intervalo de tempo curto, até que o sistema forneça algo mais próximo da resposta correta sem que eles tenham que resolver grande parte do problema por eles mesmos. Com relação à "recusa de ajuda", os alunos se recusam a pedir ajuda na forma de dicas mesmo quando eles sabem que precisam, uma vez que eles entram em um estágio da resolução do problema em que não sabem qual o próximo passo, ou simplesmente, não

sabem mais o que fazer.

Neste sentido, de acordo com Vanlehn (2006), a fim de evitar o "abuso de ajuda", alguns sistemas utilizam um esquema de pontuação para deduzir pontos ou simplesmente penalizar o aluno por cada pedido de ajuda ou dica solicitada. Já para desencorajar a "recusa de ajuda", alguns sistemas fornecem dicas quando identificam que o aluno realizou uma quantidade mínima de etapas incorretas ou simplesmente encontra-se parado há muito tempo diante da sua solução para um problema em questão. Muñoz-Merino & Kloos (2009) propõem em seu trabalho um *framework* que permite que diferentes alunos recebam diferentes dicas de acordo com o seu perfil. Neste trabalho os autores listam um série de características que, segundo os mesmos, estão em acordo com os requerimentos da modelagem de dicas. Algumas destas características são:

- Cada problema pode ter um número indeterminado de dicas relacionadas a ele;
- As dicas relacionadas a um problema podem estar disponíveis desde o início ou podem se tornar disponíveis apenas como resultado de uma resposta errada para um determinado problema;
- As dicas podem ser organizadas, em grupos, em sequência, ou mesmo seguindo uma hierarquia;
- Pode-se estabelecer um limite de apresentação de dicas aos alunos;
- Pode-se criar algum sistema de pontuação que leva em conta as dicas solicitadas e fornecidas aos alunos.

2.3.4 Utilização de Ambientes de Aprendizagem

Colocar disponível aos alunos iniciantes ferramentas e ambientes de aprendizagem que venham a facilitar o ensino de programação é uma alternativa bastante interessante. O desenvolvimento de algumas destas ferramentas e ambientes é observado, tanto a nível nacional, como por exemplo: o Portugol/Plus (Esmin, 1998), o AMBAP (Almeida et al., 2002), o AlgoLC (Petry & Rosatelli, 2006) e o VisuAlg (Souza, 2009), quanto a nível internacional, como por exemplo: o THÉTIS (Freund & Roberts, 1996), o SICAS (Mendes & Gomes, 2000), o AnimPascal (Satratzemi et al., 2001) e o BlueJ (Kölling et al., 2003).

Essa proposta é uma alternativa ao uso de Ambientes de Desenvolvimento Integrado, do inglês *Integrated Development Environment* - IDE. As IDE estão disponíveis para as linguagens de programação mais usuais e geralmente são desenvolvidas para atender as necessidades de programadores profissionais. Desta forma, para cursos introdutórios, as vantagens de se utilizar uma IDE profissional podem não ser compensadas pela sua complexidade, pois estas ferramentas possuem, muitas vezes, um extenso conjunto de conceitos e funcionalidades

que são problemáticas para alunos iniciantes, onde suas mensagens de erros e advertências podem ser de difícil compreensão, fazendo com que os alunos passem um bom tempo aprendendo a utilizar a ferramenta em si (Pears et al., 2007).

Outra estratégia interessante é a encontrada no trabalho de Bravo et al. (2005). Neste trabalho, por exemplo, os autores fazem a junção de quatro ferramentas com propósitos diferentes em um só ambiente. As ferramentas e ambientes utilizados foram: o SICAS (Mendes & Gomes, 2000), um sistema concebido para apoiar a aprendizagem dos conceitos básicos de programação estruturada, o PlanEdit (Redondo et al., 2002), uma ferramenta adaptativa para o aprendizado de design, o COLLEGE (Bravo et al., 2004), uma ferramenta que facilita o aprendizado colaborativo de programação e o OOP-Anim (Esteves & Mendes, 2004), um ambiente para a aprendizagem de conceitos básicos de Programação Orientada a Objetos.

Os autores concluíram a importância e utilidade das quatro ferramentas de forma individual, independentes umas das outras. No entanto, uma vez que elas sejam integradas em um ambiente mais amplo, permitindo uma fácil comunicação entre os diferentes instrumentos, o ambiente resultante pode ser ainda mais útil para os alunos. Esta comunicação entre as ferramentas permite, por exemplo, a animação no OOP-Anim de um código escrito colaborativamente no COLLEGE apenas pressionando um botão.

Como percebe-se, diversas ferramentas e ambientes têm sido propostos durante os últimos anos com o objetivo de facilitar o aprendizado de programação. No entanto, nota-se que, apesar de se encontrarem relatos de resultados positivos com a utilização de algumas destas ferramentas, a verdade é que nenhuma delas tem uma utilização generalizada.

Para Pears et al. (2007), há várias razões para isso. Primeiramente, a maioria destas ferramentas é oriunda de projetos de pesquisa e muitas vezes elas são concebidas com a intenção de resolver um problema local, em uma instituição específica ou em um curso específico. Além disso, as diferenças locais tornam difíceis possíveis adaptações nestas ferramentas.

Uma segunda razão é consequência do processo de investigação. Estas ferramentas são muitas vezes desenvolvidas utilizando recursos de pesquisa ou como parte de uma pesquisa de iniciação científica, mestrado ou doutorado, não sendo o objetivo principal da pesquisa e sim apenas um protótipo para validação.

2.3.5 Utilização de Tutores/Assistentes Inteligentes

Para Pillay (2003), a utilização de tutores de programação inteligentes é um meio eficaz e economicamente viável de ajudar programadores iniciantes a superar as dificuldades de aprendizagem. No entanto, a utilização generalizada destes tutores tem sido inibida pelos elevados custos de desenvolvimento destes sistemas. Infelizmente, apesar das diversas pesquisas na área, a ideia de um STI em que o aluno receba auxílio personalizado, conforme suas necessidades específicas, ainda está distante do que realmente se poderia e se gostaria

de fazer.

Para Giraffa (1999), as razões para tais limitações residem tanto nos aspectos de *hardware* e *software*, quanto nos aspectos psicológicos e pedagógicos. Além disso, pesquisas nas áreas de psicologia e educação não oferecem ainda teorias computáveis em que se possam aplicar e realizar simulações mais adequadas. O que existe são adaptações e combinações de soluções geradas pelas três grandes áreas, a fim de se melhorar os ambientes computadorizados de ensino-aprendizagem.

Neste sentido, tendo noção de que a realidade de desenvolvimento dos STI está muito distante de simular um comportamento o mais próximo possível ao de um professor humano, novas alternativas estão sempre sendo desenvolvidas.

A utilização de assistentes inteligentes menos complexos, porém não menos eficazes, parecer ser uma alternativa interessante para suprir as dificuldades no desenvolvimento dos STI, como a proposta no trabalho de Chou et al. (2003), onde a inteligência de máquina e a inteligência humana devem complementar-se. Ou seja, o sistema pode executar algumas tarefas de tutoria, enquanto o professor humano executa as tarefas restantes, ou o sistema pode prover tutoria apenas a alguns alunos enquanto o professor auxilia o restante dos alunos.

Sendo assim, uma das formas de auxiliar aos alunos acontece no momento em que o professor também recebe algum tipo de auxílio para que possa transmitir seu conhecimento aos alunos de forma mais eficiente. A partir desta perspectiva, os STI podem ser subdivididos em algumas subcategorias. De acordo com Yacef (2002), ao classificar os sistemas inteligentes de ensino de acordo com seu grau de autonomia para o professor, as seguintes classes de sistemas são encontradas:

1. **Os que são totalmente autônomos e substituem o professor**;
2. **Os que auxiliam o professor, mas possuem autonomia de uso** - O aluno pode praticar sem a presença do professor;
3. **Os que auxiliam o professor e não são autônomos** - O professor deve permanecer presente durante a interação do aluno com o sistema.

Considerando que grande parte da literatura sobre os STI está focada nos sistemas que se enquadram na classe 1, Yacef (2002) propôs distinguir as outras duas classes, as quais enquadram-se os Assistentes Inteligentes de Ensino, do inglês *Intelligent Teaching Assistant* - ITA.

Os ITA são sistemas que possuem o objetivo de auxiliar, de forma inteligente, não somente os alunos, mas também os professores, reduzindo ou automatizando tarefas tediosas, além de disponibilizar auxílio e assistência individualizada aos alunos, facilitando também a disponibilização de materiais e exercícios personalizados. (Yacef, 2002)

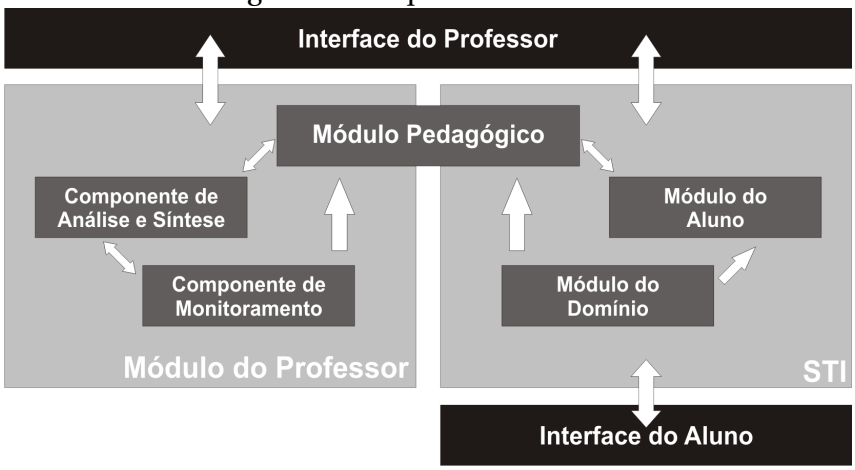
Além do motivo de prover um foco maior no papel dos professores neste processo, esta subdivisão também se justifica, de acordo com Pinto (1995), em função do porte e complexidade destes sistemas. Enquanto os ITA realizam tarefas mais simples, com um modelo de aluno simplificado e estratégias pedagógicas restritas baseadas no monitoramento do aluno e nos resultados obtidos ao longo da interação, os STI possuem modelos mais complexos e estratégias pedagógicas mais invasivas, apresentando um maior número de interrupções do tutor na interação com o aluno.

2.3.5.1 Arquitetura dos Assistentes Inteligentes de Ensino

De acordo com Pinto (1995), os Assistentes Inteligentes de Ensino, vistos como um subconjunto dos Sistemas Tutores Inteligentes, teriam os mesmos componentes, assim como, a mesma arquitetura básica destes sistemas.

No entanto, por possuir dois usuários-alvo diferentes (aluno e professor), os ITA apresentam uma arquitetura um pouco diferente, conforme pode ser observado na Figura (2.1). Em um ITA o professor humano compartilha as estratégias pedagógicas com o STI, auxiliado pelas ferramentas do módulo do professor, podendo interferir e influenciar no processo decisório. Assim, a figura do professor permanece presente no controle do processo de ensino/aprendizagem (Yacef, 2002).

Figura 2.1: Arquitetura dos ITA.



Fonte: Adaptado de Yacef (2002).

O Módulo do Professor e sua respectiva interface são as principais diferenças em comparação com os STI tradicionais. Porém, segundo Yacef (2002), este módulo pode variar de acordo com o objetivo do ITA. Além dos módulos do aluno, domínio, estratégias pedagógicas e interface, a arquitetura de um ITA possui o módulo do professor, composto por um componente pedagógico, um componente de monitoramento e um componente de análise e síntese, além da sua respectiva interface por onde o professor interage com o sistema.

O Módulo Pedagógico, já presente na arquitetura clássica de um STI, é aberto para permitir a entrada do professor. Este componente contribui para a oferta de material pedagógico, como por exemplo: geração de material adaptado (exercícios, perguntas, comentários, explicações, etc).

O Componente de Monitoramento contém o conhecimento e os procedimentos necessários para acompanhar os alunos ou um grupo de alunos durante a interação com o sistema. Este componente pode ser útil para armazenar e apresentar análise ao professor sobre incidentes ocorridos durante a interação do aluno com o sistema.

O Componente de Análise e Síntese inclui o conhecimento e os procedimentos de análise (identificação de tendências e áreas de problema, tanto a nível individual, quanto a nível de grupo) e síntese (compilação e visualização de resultados, tanto a nível individual, quanto a nível de grupo, sob vários critérios e tipos de apresentação).

Cada um destes módulos pode aumentar a complexidade da interface, dependendo de quão ativo seja o papel do professor: relativamente passivo (apenas visualizar resultados), moderadamente ativo (aceitar ou não as sugestões feitas pelo sistema sobre próximos exercícios ou treinamentos), ou muito ativo (modificar dados de diagnóstico do aluno, modificar objetivos de treinamento, etc). A interface do professor deve ter acesso a ambos os módulos, tanto do professor, quanto do STI, uma vez que o professor deve ser capaz de modificar dados no STI (informações no módulo do aluno, por exemplo).

O módulo do STI contém os componentes tradicionais geralmente aceitos na literatura destes sistemas, com algumas variações introduzidas pela presença do professor.

2.3.5.2 Vantagens de um Assistente Inteligente de Ensino

De acordo com Yacef (2002), os ITA podem oferecer diversas formas de assistência, tais como:

- **Auxiliar no diagnóstico e avaliação da aprendizagem**. Os ITA podem ajudar na elaboração de um diagnóstico de aprendizagem mais preciso, sistêmico e/ou mais rápido.
- **Auxiliar na geração de materiais sob medida**. Os ITA podem ajudar na criação ou adaptação de exercícios/questões, tendo em vista informações sobre o aprendizado e dificuldades do aluno.
- **Auxiliar no monitoramento do aluno durante a execução de um exercício**. Os ITA podem fornecer um acompanhamento atento do aluno durante a execução de um exercício, liberando o professor desta tarefa, reportando à ele os resultados de forma significativa.
- **Auxiliar na análise ou síntese de resultados**. Os ITA podem coletar dados dos alunos (interações com o sistema) e comunicar ao professor de várias formas.

Além destes tipos de assistência, há ainda duas questões que os ITA podem ter como objetivo:

- **Reduzir a quantidade de tarefas onerosas que podem ser automatizadas ou facilitadas**. Se a carga de trabalho do professor é reduzida, consequentemente ele estará mais disponível para cumprir tarefas onde sua experiência é importante.
- **Melhorar a qualidade do processo de ensino, fornecendo ao professor novas/melhores ferramentas, além de *feedback***. Se o professor possui uma melhor visão sobre a aprendizagem dos alunos e formas de prover um melhor material de ensino, consequentemente ele pode oferecer um ensino melhor e mais focado aos seus alunos.

Como é possível perceber, a filosofia de um ITA é considerar o professor também como um usuário final do sistema, ao invés de substituí-lo totalmente. Para Yacef (2002), embora não exista nenhuma limitação óbvia do tipo de domínio onde os ITA podem ser utilizados, seus benefícios são mais notáveis em domínios onde o professor é um recurso escasso.

Neste trabalho utiliza-se um assistente inteligente que auxilia os alunos no processo de ensino/aprendizagem de programação através da resolução de problemas. Esta escolha foi uma alternativa pelo reconhecimento das dificuldades de construção de um STI completo e mais complexo que atenda aos requisitos esperados por este tipo de sistema.

Outro ponto é o reconhecimento do papel do professor neste processo. É válido acreditar que, principalmente no domínio da programação de computadores, um sistema totalmente autônomo e que substitua o papel do professor talvez não atenda as expectativas dos alunos com relação ao desenvolvimento de seu aprendizado, devido principalmente ao caráter subjetivos de alguns aspectos avaliativos, tais como: proximidade da resposta correta, estilo de programação, criatividade, maturidade do aluno, capacidade de compreensão e resolução de problemas, etc. Sendo assim, a ideia de utilizar um assistente inteligente que auxilie também o professor em algumas de suas tarefas apresenta-se como uma alternativa bastante interessante no nosso contexto.

3 AMBIENTES E FERRAMENTAS VOLTADOS AO ENSINO DE PROGRAMAÇÃO

Os ambientes de programação profissionais, ou de desenvolvimento integrado, são uma fonte de obstáculos para os alunos, principalmente alunos iniciantes no domínio da programação. Estes ambientes, também conhecidos como IDE - *Integrated Development Environment*, geralmente não fornecem a simplicidade de uso que os alunos iniciantes precisam.

Uma discussão mais aprofundada sobre este assunto pode ser encontrada no trabalho de Gómez-Albarrán (2005), onde a autora sintetiza diferentes direções e abordagens para a melhoria do processo de ensino/aprendizagem de programação através da apresentação, análise e discussão de diversos ambientes e ferramentas voltados para o ensino de programação. Cerca de 20 ambientes e ferramentas descritos na literatura e utilizados em diferentes instituições de ensino no mundo são analisados. Através do estudo das suas características e funcionalidades, a autora conseguiu classificá-las em quatro categorias:

1. **Ambientes/Ferramentas de desenvolvimento reduzido**;
2. **Ambientes/Ferramentas baseados em exemplos**;
3. **Ambientes/Ferramentas baseados em visualização e/ou animação**;
4. **Ambientes/Ferramentas de simulação**.

A seguir são apresentados alguns dos ambientes e ferramentas analisados por Gómez-Albarrán (2005), além de outros encontrados na literatura, de acordo com a classificação proposta por essa autora. São listadas algumas de suas respectivas características e como estes sistemas podem auxiliar para o desenvolvimento do processo de ensino/aprendizagem de programação.

3.1 Ambientes/Ferramentas de Desenvolvimento Reduzido

Estes ambientes/ferramentas possuem recursos simples para o desenvolvimento de algoritmos, onde os alunos podem encontrar facilidades para escrever, compilar, executar e depurar os seus códigos. Alguns destes ambientes costumam possuir também características de diagnóstico.

Um dos primeiros e mais representativos ambientes enquadrados nesta categoria é o THÉTIS (Freund & Roberts, 1996). O THÉTIS contém recursos de edição, compilação e execução de programas escritos utilizando a linguagem de programação C. Com relação ao editor de código, o THÉTIS não fornece opção de formatação automática de código, como

encontrado normalmente em editores profissionais. Porém, inclui um sistema de diagnóstico de erros de execução, oferecendo *feedbacks* sobre as causas e a localização de diversos erros de execução, como por exemplo, divisão por zero, uso de variáveis não inicializadas, acesso a matrizes fora de seus limites, passagem de argumentos inválidos para funções, dentre outros.

Outro exemplo é o AnimPascal (Satratzemi et al., 2001), um ambiente atraente para o ensino da linguagem de programação Pascal. O AnimPascal possui as características típicas de edição e compilação de código, juntamente com a visualização da execução do programa.

Outro exemplo, atualmente utilizado em diferentes universidades no mundo, é o ambiente BlueJ (Kölling et al., 2003), uma continuação da linguagem e do ambiente Blue (Kölling & Rosenberg, 1996). O BlueJ é um ambiente de desenvolvimento integrado Java que foi criado para apoiar o ensino de programação através da abordagem de objetos. As versões mais recentes do BlueJ possuem alguns recursos, como: exercícios e tarefas, demonstrações, tutoriais e documentação.

Outro ambiente que se enquadra nesta categoria é o DrJava (Allen et al., 2002), um ambiente de desenvolvimento Java criado na Universidade de Rice. O editor do DrJava suporta identação automática, destaque de palavras-chave, além de eliminar a dependência do console de I/O Java.

O SICAS (Mendes & Gomes, 2000) é um ambiente que permite que os alunos desenvolvam seus algoritmos e resolvam problemas de programação dando pouca atenção aos detalhes sintáticos, com base na experimentação e na prática. A construção da resolução dos problemas é realizada através de fluxogramas, o que privilegia o uso de representações gráficas, ao invés de representações textuais (pseudocódigos). Na verdade, o SICAS disponibiliza tanto o ambiente visual com fluxogramas, como o textual, com pseudocódigo, permitindo com isso que os alunos possam escolher qual das opções utilizar, apesar do foco da ferramenta ser a construção das soluções na forma de fluxogramas.

O Portugol/Plus (Esmin, 1998), é uma ferramenta desenvolvida na linguagem Pascal para a plataforma DOS com o objetivo de estimular o aprendizado de lógica de programação utilizando o Portugol. A ferramenta apresenta um editor baseado nos editores de texto da plataforma DOS, permitindo a manipulação dos algoritmos, e um compilador de algoritmos formando por um analisador léxico e um analisador sintático, que gera ao final, um programa em Pascal.

O CGRAPHIC (Fernández & Sánchez, 2003) é um ambiente interativo e gráfico que atua como tutor virtual para ensinar fundamentos de programação, podendo ser executado a partir de qualquer navegador. O CGRAPHIC oferece dois tipos de aprendizagem: teórico e prático. No *framework* teórico, o CGRAPHIC fornece explicações teóricas *on-line* na forma semelhante à de um livro clássico de programação. O *framework* prático está associado a um conjunto de exercícios interativos, onde cada exercício requer a implementação de um programa estruturado em C que trata ou que está relacionado com o tema atual do *framework* teórico.

O AMBAP (Almeida et al., 2002) é um ambiente desenvolvido para auxiliar iniciantes no processo de ensino de programação. O ambiente dispõe de um editor de código, um interpretador e um tradutor, permitindo ao aluno desenvolver e executar seus programas utilizando uma linguagem algorítmica simples sem se preocupar com detalhes inerentes à implementação, comumente encontrados nas linguagens de programação convencionais.

3.2 Ambientes/Ferramentas Baseados em Exemplos

Os seres humanos usam as soluções de problemas anteriores como base para resolver novos problemas semelhantes. Isto também se aplica no domínio da programação, onde tanto especialistas quanto programadores iniciantes utilizam programas já escritos ou aprendidos, a fim de resolver novos problemas.

Desta forma, alguns ambientes de programação baseados em exemplos têm sido desenvolvidos. Segundo Gómez-Albarrán (2005), estes ambientes incluem uma base de exemplos de programação e mecanismos que extraem exemplos relevantes e reutiliza-os para resolver novos problemas.

O ELM-PE (Weber, 1996) é um ambiente de ensino baseado em exemplos que auxilia o aprendizado da linguagem de programação LISP. Os alunos trabalham na resolução dos problemas enquanto o ELM-PE permanece passivo e oferecendo *feedback* sob demanda. O ELM-PE incentiva os alunos a reutilizarem exemplos fornecidos no material de aprendizagem inicial do ambiente, bem como, aqueles previamente construídos pelos alunos.

O ELM-ART (Weber & Brusilovsky, 2001) é um ambiente baseado no ELM-PE, mas voltado para a Web. O ELM-ART também auxilia o aprendizado da linguagem de programação LISP e pode ser visto como um livro interativo inteligente representado eletronicamente na forma de hipertexto. Os conteúdos são hierarquicamente estruturados em unidades de diferentes níveis: capítulos, seções e subseções. O ELM-ART é composto por mecanismos de seleção de vários exercícios dirigidos ao aluno, baseados em uma ferramenta de pesquisa. Uma característica adicional é a geração de explicações de erros em diferentes níveis, desde dicas vagas até explicações mais completas.

Outra ferramenta interessante que se enquadra nesta categoria é o WebEx (Brusilovsky, 2001), uma ferramenta baseada na Web que explora interativamente exemplos de programação. Uma característica inovadora neste sistema é a utilização de exemplos auto-explicáveis ao invés de exemplos de códigos de programação puros. O WebEx contém uma base de exemplos de programação com explicações textuais fornecidas a cada linha de código, essas explicações informam o significado de cada linha de código e seu papel na solução global. Outra característica interessante do WebEx é a sua capacidade de gravação das ações do aluno no ambiente, desta forma, o instrutor pode monitorar a atividade estudantil podendo chegar a algumas conclusões sobre a forma como os alunos trabalham.

3.3 Ambientes/Ferramentas Baseados em Visualização e/ou Animação

Para Gómez-Albarrán (2005), o fato dos seres humanos serem bons no processamento visual das informações reflete com precisão a enorme capacidade de comunicação as imagens proporcionam. Estes dois fatos, juntamente com a ascensão das interfaces modernas humano-computador, levaram ao uso de softwares de visualização como ferramenta pedagógica no processo de ensino de programação (Gómez-Albarrán, 2005).

Esta abordagem vem sendo pesquisada há algum tempo e consiste no uso da computação gráfica e da animação para auxiliar, ilustrar e apresentar processos, algoritmos e programas de computador. Esta abordagem pode ser utilizada no ensino para auxiliar os alunos a entenderem como os algoritmos funcionam e também pode ser utilizada para auxiliar os programadores a compreenderem melhor o seu código.

O nascimento do campo da animação de algoritmos pode estar ligado à apresentação do filme ***Sorting Out Sorting***[1] (Baecker & Sherman, 1981), na conferência SIGGRAPH da ACM - *Association for Computing Machinery* em 1981. O filme apresenta uma animação de dados com uma narrativa explicativa para introduzir nove diferentes algoritmos de ordenação interna.

Gómez-Albarrán (2005) afirma que, além das animações desenvolvidas, inúmeros ambientes e ferramentas para produzir ou apresentar animações de algoritmo estão disponíveis. Estes ambientes e ferramentas diferem na forma como as animações são geradas, nos recursos de animação oferecidos, nos aspectos de portabilidade e nos fatores de custo. Diversas abordagens para a geração de animações de algoritmo podem ser descritas: edição visual, animação direta do código-fonte, animação de chamadas de função, produção declarativa e uso de linguagem de *script*. Segundo Gómez-Albarrán (2005), alguns destes ambientes e ferramentas disponíveis utilizam mais de uma abordagem para a produção de animações ou utilizam uma combinação especial de algumas abordagens.

O LEONARDO (Crescenzi et al., 2000), por exemplo, é um ambiente desenvolvido para escrever e animar programas na linguagem C. O ambiente possui um completo controle da exibição da animação. O código de animação no LEONARDO também pode ser compilado em outros ambientes.

O JHAVÉ (Naps et al., 2000) é um ambiente cliente-servidor que suporta a visualização de algoritmos em três diferentes linguagens de *script*: Samba, Animal e Gaigs, além de ser compatível com algumas exigências didáticas. Os alunos podem contribuir provendo entradas para a execução do algoritmo. Além disso, o JHAVÉ força o aluno a participar ativamente do processo interrompendo a visualização dos algoritmos com perguntas, onde os alunos devem prever o que acontecerá na próxima etapa da animação, através da aplicação de testes de

[1] Também disponível em: `http://video.google.com/videoplay?docid=3970523862559774879`

múltipla escolha.

Para Ala-Mutka (2004), a maioria dos ambientes e ferramentas enquadrados nesta categoria, se concentram na animação de algoritmos e dão menos ênfase para a visualização da estrutura básica dos programas e a sua execução. Para o autor, no entanto, há resultados que podem ser utilizados também na concepção de abordagens para visualizar as estruturas básicas de programação.

Esta ideia é confirmada por Gómez-Albarrán (2005), que afirma a existência de outros tipos de ambientes e ferramentas baseados em visualização, os que utilizam a abordagem de "ensino de máquina". Estes ambientes e ferramentas apresentam visualmente o que está acontecendo dentro da máquina (estado da memória e outros recursos), quando um programa é executado, ou seja, enquanto alguns omitem o código ou disponibilizam-no na forma de pseudocódigo, como o JHAVÉ por exemplo, outros visualizam a estrutura interna e a execução do código.

A família JELIOT é um exemplo deste tipo de ambiente. O ELIOT (Lahtinen et al., 1998) foi desenvolvido para visualizar e animar programas em C, já o JELIOT I (Haajanen et al., 1997) e seu sucessor JELIOT 2000 (Levy et al., 2003), auxiliam os alunos através da animação e visualização de um subconjunto da linguagem Java. O JELIOT, em sua versão mais atual, oferece um ambiente de visualização completo com a capacidade de "dar vida" aos programas, ou seja, animá-los, permitindo a inserção de código que será visto, literalmente, da mesma forma que o computador interpreta e executa as ações.

Outro sistema que se enquadra nesta categoria é o Jpie (Goldman, 2004), que suporta o desenvolvimento interativo de software orientado a objetos em Java. No Jpie os alunos manipulam diretamente as representações gráficas para efetuar mudanças no código em execução. Muitas operações, tais como declarações de variáveis e de métodos são realizadas através de operações do tipo *drop-down* (arrastar e soltar).

Enquanto em um ambiente de programação tradicional as alterações feitas no código requerem um passo para recompilação, ou seja, reiniciar o aplicativo e em seguida executá-lo novamente para ver o resultado da mudança no código, no Jpie, este ciclo de editar-compilar-testar é eliminado. Ao invés disto, os alunos manipulam diretamente o código enquanto eles estão funcionando e podem ver seus efeitos imediatamente.

De acordo com Goldman (2004), esse *feedback* rápido não só poupa um tempo considerável, mas também cria uma situação na qual os alunos iniciantes são incentivados a aprender através da experimentação. Além disso, o JPie fornece uma opção onde o aluno pode visualizar a relação entre a representação gráfica especificada e sua correspondente versão textual. Ao fornecer este recurso, o ambiente tem o cuidado em gerar um código correspondente aceitável para um aluno em um curso introdutório.

No estudo realizado por Rössling & Freisleben (2000), os autores afirmam que os alunos que interagem com animações e visualizações de algoritmo possuem um desempenho significativamente melhor do que aqueles que apenas ouvem o professor, além de afirmarem que

esta metodologia, baseada em animação e visualização de algoritmos, aumenta a motivação dos alunos.

No entanto, segundo Gómez-Albarrán (2005), para ser eficaz, a animação de algoritmos deve extrair e ressaltar apenas aspectos essenciais dos algoritmos, onde os desenvolvedores destas ferramentas devem ficar atentos no sentido de conceber desenhos gráficos atraentes e organizados, além de empregar os ritmos e passos das animações de forma apropriada.

Para Ala-Mutka (2004), uma crítica contra esta abordagem é a quantidade de tempo necessário para criar, instalar e estudar a tecnologia envolvida nestas visualizações. Entretanto, para o autor, os professores devem sempre lembrar que estas ferramentas e técnicas de visualizações e animações são apenas auxiliadoras no processo de ensino/aprendizagem, e que outras questões devem ser consideradas ao se utilizar esta abordagem de ensino.

3.4 Ambientes/Ferramentas de Simulação

Os ambientes e ferramentas de simulação têm como objetivo ajudar os alunos a compreender os programas e os seus efeitos por meio de simulações. Ao contrário dos ambientes e ferramentas baseados em visualização e animação, os ambientes e ferramentas de simulação refletem a execução de um programa em um mundo imaginário (Gómez-Albarrán, 2005).

Um dos mais bem sucedidos ambientes de simulação voltado para o ensino de programação talvez seja o Karel the Robot (Pattis, 1981). Durante muito tempo o Karel foi usado para introduzir alunos iniciantes aos princípios fundamentais de programação estruturada.

Karel é um robô que vive em um mundo 2D, onde as coordenadas horizontais são chamadas de ruas que vão de leste a oeste e as coordenadas verticais são chamadas de avenidas e vão de norte a sul. As ruas e avenidas são enumeradas por números inteiros positivos. Existem paredes impenetráveis em algumas ruas e avenidas, não podendo existir paredes situadas entre cruzamentos.

Os robôs são equipados com vários dispositivos e possuem funcionalidades, como: ir para frente, virar à esquerda 90º, virar à direita 90º ou finalizar alguma tarefa específica. Além do conjunto reduzido de ações, correspondentes ao conjunto reduzido de instruções da linguagem de programação usada para escrever o programa. Os alunos podem definir suas próprias funções implementadas usando a linguagem de programação Karel, onde o robô pode testar condições para selecionar alternativas ou ações de repetição.

Além de seus descendentes diretos, o Karel the Robot inspirou outros ambientes de simulação. O JKarelRobot (Buck & Stucki, 2001), por exemplo, auxilia o ensino das linguagens de programação Pascal, Java e Lisp. Algumas das características que diferenciam o JKarelRobot da "família Karel" estão ligadas às suas melhorias na forma de execução dos códigos. O JKarelRobot permite não só um modo de execução contínua, mas também, um modo de execução passo-a-passo com recursos que permitem o programa desfazer um passo. Além

disso, o código do programa também é destacado linha por linha no editor a fim de indicar a declaração que está atualmente sendo executada.

Outro ambiente não relacionado com a "família Karel", é o Alice (Cooper et al., 2003). O Alice é um ambiente 3D interativo onde os alunos escrevem seus programas através de uma linguagem semelhante à linguagem Java, usando um editor para arrastar e soltar ou para selecionar os habitantes do mundo e programar seus comportamentos. Os alunos são, de certa forma, protegidos pelo ambiente contra os erros de sintaxe, uma vez que eles só podem arrastar e soltar os componentes do programa em locais sintaticamente corretos.

3.5 Considerações Finais Sobre os Ambientes e Ferramentas Voltados ao Ensino de Programação

Foram apresentados neste capítulo, de forma não exaustiva, alguns dos diversos ambientes e ferramentas existentes na literatura que possuem o objetivo de auxiliar no processo de ensino/aprendizagem de programação, seguindo a classificação proposta por Gómez-Albarrán (2005).

Com relação à primeira classe destes ambientes e ferramentas, é possível chegar à conclusão de que estes são adequados, principalmente, para os alunos iniciantes. Para tanto, Gómez-Albarrán (2005) identificou em seu trabalho algumas das principais características destes ambientes e ferramentas, que consequentemente os tornam adequadas para utilização com alunos iniciantes, são elas: a linguagem utilizada para escrever os códigos, a forma de edição do código, o processo de compilação, os recursos de depuração, os erros de execução verificados e explicados pela ferramenta, além de outros recursos adicionais.

Já os ambientes e ferramentas baseados em exemplos são caracterizados pelo uso extensivo de exemplos de programação como ponto de partida para resolver novos problemas. Com relação a esta categoria, Gómez-Albarrán (2005) analisou em seu trabalho a granularidade dos exemplos, a abordagem de seleção de exemplos seguida, se o ambiente pede ao aluno para resolver os exercícios, o tipo de apoio provido na resolução dos exercícios, se os alunos tem acesso aos exercícios resolvidos, dentre outras características. Esta abordagem pode ser considerada como uma forma natural de aprendizagem, no entanto, para Gómez-Albarrán (2005), o sucesso destes ambientes ainda não é tão forte como o esperado, uma vez que os alunos podem encontrar dificuldades ao procurar exemplos relevantes para seu aprendizado. Uma alternativa útil é interessante seria a navegação entre os exemplos baseada na similaridade ou algum mecanismo de busca para encontrar exemplos semelhantes ao que já foram explorados.

Com relação aos ambientes e ferramentas baseados em visualização e/ou animação, entende-se que estes facilitam o processo de ensino de programação através da exibição do comportamento do código. A análise de Gómez-Albarrán (2005) relacionada a estes

ambientes e ferramentas focou nos seguintes fatores: geração das animações, recursos de animação oferecidos e aspectos de portabilidade e de custo. De acordo com Gómez-Albarrán (2005), muitos professores da área de computação estão convencidos que estes ambientes e ferramentas são bastante atraentes, uma vez que os alunos são bastante atenciosos ao processamento visual das informações, prendendo sua atenção no aprendizado dos conceitos de forma mais efetiva.

Para a análise dos ambientes e ferramentas de simulação, Gómez-Albarrán (2005) considerou os seguintes aspectos: tipo de mundo onde os personagens vivem, comportamento de acordo com o código do aluno, linguagem utilizada para escrever os códigos, tipo de editor utilizado para escrever os códigos, comandos de ação disponíveis para alterar o estado dos habitantes do mundo, tipos de simulação gerados, características de tempo de execução, dentre outros aspectos. Após esta análise, é possível concluir que o fato de alguns alunos da área da computação ligarem a computação aos jogos eletrônicos, faz com que estes ambientes e ferramentas de simulação pareçam uma alternativa bastante atraente para o aprendizado, uma vez que a execução do código é refletida em um mundo imaginário e nas ações de seus habitantes.

Independente de qual categoria um ambiente ou ferramenta se enquadre, é importante ressaltar que cada um tem a sua contribuição individual. Encontrar um ambiente ou ferramenta voltado para o ensino de programação que englobe todas as características e funcionalidades desejadas ainda é algo difícil.

A proposta de ambiente apresentada neste trabalho com certeza não possui todas as funcionalidades e características que atendam as expectativas esperadas pela comunidade da área, assim como pelos alunos. No entanto, acredita-se que ela possui sua contribuição por englobar funcionalidades e características presentes em mais de uma categoria, de acordo com a classificação proposta por Gómez-Albarrán (2005).

Tabela 3.1: Principais características das categorias de Ambientes/Ferramentas propostas por Gómez-Albarrán (2005).

Categorias segundo Gómez-Albarrán (2005)	**Principais características de cada categoria**	**AIIP**
Ambientes/Ferramentas de desenvolvimento reduzido	• Simplicidade da linguagem utilizada.	3
	• Facilidade de edição e compilação/interpretação do código.	3
	• Recursos de depuração.	2
	• Erros de execução verificados e explicados pela ferramenta.	2
Ambientes/Ferramentas baseados em exemplos	• Diversidade dos exemplos oferecidos.	2
	• Incentivo à resolução dos exercícios/problemas.	3
	• Apoio na resolução dos exercícios/problemas.	3
	• Acesso aos exercícios/problemas resolvidos.	2
Ambientes/Ferramentas baseados em visualização e/ou animação	• Geração de animação.	2
	• Diversidade de recursos de animação.	1

Fonte: Próprio autor.

Como é possível observar na Tabela 3.1, foram listadas algumas das características mais marcantes de cada categoria, atribuindo-se uma escala com o objetivo de quantificar a incidência destas características no ambiente proposto neste trabalho, onde: 1 = Pouco, 2 = Razoável e 3 = Muito. A categoria Ambientes/Ferramentas de simulação não foi incluída nesta tabela pelo fato do ambiente proposto neste trabalho não englobar nenhuma característica desta categoria.

Os próximos dois capítulos são dedicados exclusivamente à apresentação e detalhamento do ambiente proposto.

4 AIIP – AMBIENTE INTELIGENTE PARA INICIANTES EM PROGRAMAÇÃO (CARACTERÍSTICAS E DESCRIÇÃO DA ARQUITETURA)

O AIIP - Ambiente Inteligente para Iniciantes em Programação (Figuras 4.1 e 4.2) é um ambiente cujo objetivo é auxiliar no processo de ensino/aprendizado de programação de alunos iniciantes. Utiliza-se no AIIP uma abordagem de resolução de problemas, apoiada por um assistente inteligente que auxilia os alunos durante a interação com o ambiente, fornecendo dicas e *feedbacks* apropriados e avaliando de forma automática suas soluções algorítmicas de acordo com métricas pré-estabelecidas.

Figura 4.1: Tela de *login* do AIIP.

Fonte: Próprio autor.

O desenvolvimento do AIIP, incluindo a sua arquitetura, foi realizado a partir do AMBAP (Almeida et al., 2002). A linguagem de programação utilizada no AMBAP, e consequentemente adotada no AIIP, é baseada no ILA [1] Evaristo & Crespo (2003), que possui propósito educacional e cujas instruções são baseadas na língua portuguesa, ou seja, aumentando o foco na lógica de programação em si e diminuindo a complexidade de aprendizado por parte dos alunos quando comparados com o uso de linguagens como C e Java.

[1] professor.unisinos.br/wp/crespo/ila/

Figura 4.2: Tela do Menu Principal do AIIP.

Fonte: Próprio autor.

Porém, diferentemente do AMBAP, o AIIP possui um aspecto mais educacional. Além do ambiente prático, é disponibilizado também no AIIP um ambiente teórico de ensino, onde os alunos podem aprender através da navegação entre os conceitos e exemplos sobre lógica de programação.

São utilizadas também no AIIP algumas estratégias pedagógicas de aprendizagem, como por exemplo: a apresentação de dicas gerais e específicas durante a resolução dos problemas, a apresentação de *feedbacks* a cada erro cometido e a apresentação de informações estatísticas específicas e gerais referentes ao desempenho dos alunos.

Além destas funcionalidades, o AIIP possui um assistente inteligente que acompanha o aluno durante sua interação com o ambiente, monitorando-o e, quando necessário, interrompendo-o para informá-lo sobre algum aspecto relevante em seu aprendizado.

Outra característica interessante é a flexibilidade tanto na forma de avaliação dos alunos, quanto nos conteúdos e problemas armazenados no ambiente, que podem ser modificados pelo professor a depender de aspectos como por exemplo: nível de conhecimento da turma ou conceitos estudados em um dado momento.

Ou seja, além de prover um assistente inteligente para auxiliar os alunos durante a resolução dos problemas e avaliar as soluções dos mesmos de forma automática, é possível acreditar que a proposta aqui apresentada atende a alguns dos requisitos esperados pelos ambientes e ferramentas que se propõe a auxiliar no processo de ensino/aprendizagem de programação.

No decorrer deste capítulo as funcionalidades e os componentes do AIIP serão apresentados, juntamente com seus objetivos e a forma como são utilizados e ativados pelos professores/tutores, pelos alunos ou mesmo de forma automática.

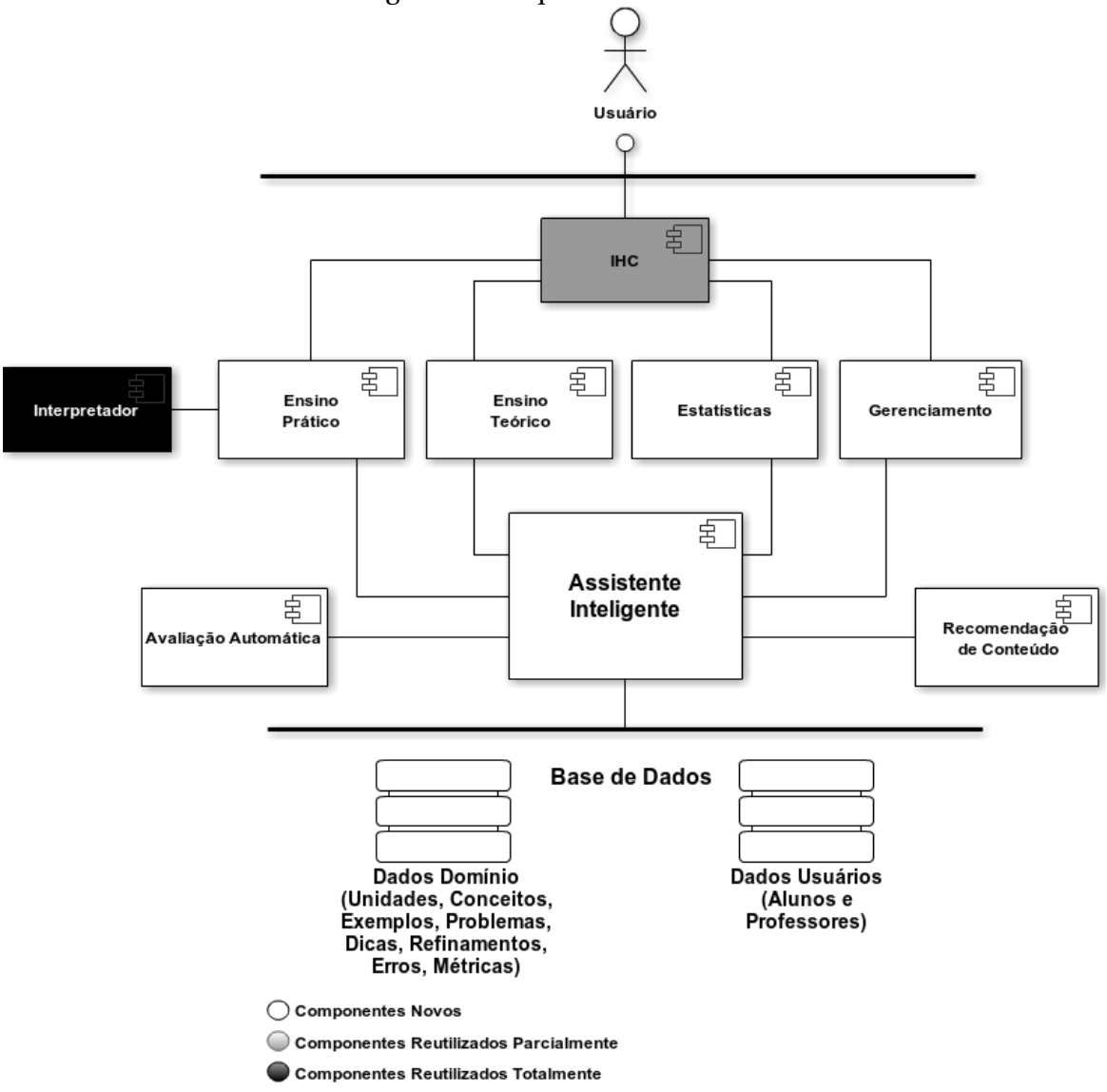
4.1 Arquitetura do AIIP

A primeira arquitetura proposta para o AIIP, dividida em quatro módulos principais de acordo com o estilo arquitetural em camadas de Shaw & Garlan (1996), inicialmente apresentada no trabalho de Lima et al. (2011), precisou ser modificada para que novas funcionalidades fossem adicionadas, e principalmente para centralizar no assistente inteligente o comando das principais atividades do sistema.

Como comentado anteriormente, alguns componentes foram reutilizados e adaptados do AMBAP, foram eles: o Editor e o Simulador (que agora pertencem a um único componente, o IHC) e o Interpretador (consistente de uma biblioteca externa baseada no ILA).

A arquitetura proposta está ilustrada na Figura 4.3. A funcionalidade de cada componente da arquitetura é explicada a seguir.

Figura 4.3: Arquitetura do AIIP.



Fonte: Próprio autor.

4.1.1 Interação Humano Computador (IHC)

É o componente responsável por apresentar as telas do sistema e acionar, quando oportuno, os demais componentes. Caso o usuário seja um aluno, ele poderá acessar os ambientes de ensino teórico e prático, e o ambiente de estatísticas. No caso do professor, além destes ambientes, ele ainda dispõe de um ambiente responsável por gerenciar o conteúdo pedagógico do AIIP, além das métricas de avaliação e dos níveis de aprendizado dos alunos.

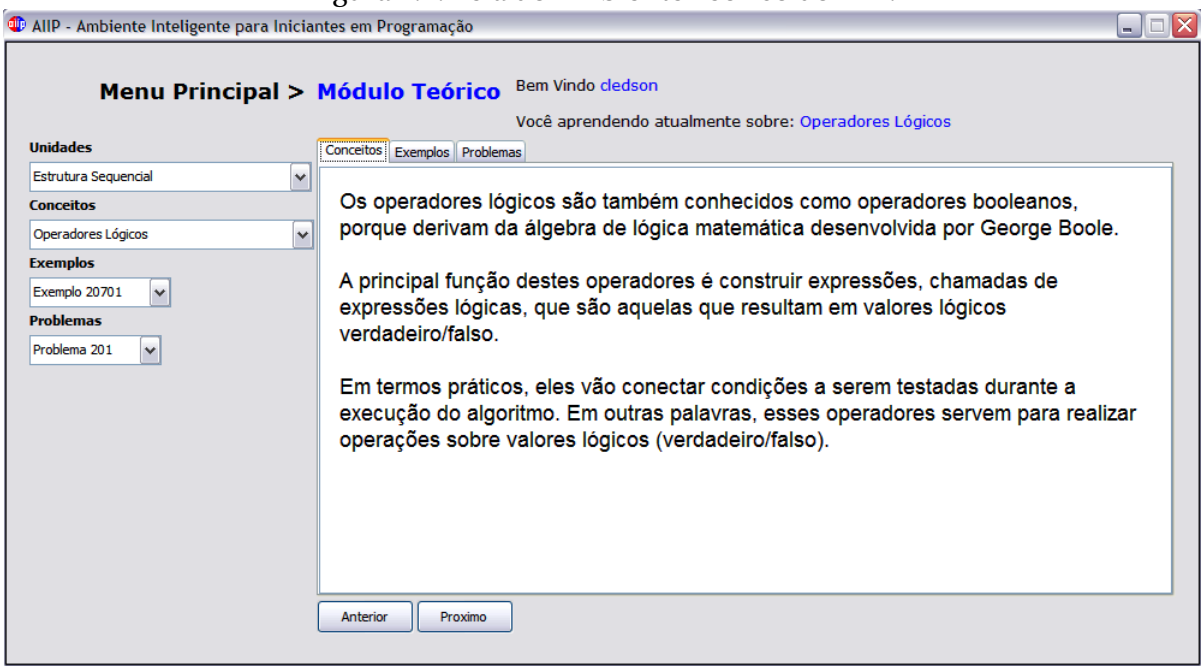
4.1.2 Interpretador

É o componente responsável por executar as instruções definidas pelo usuário em seus algoritmos. Consiste de uma biblioteca externa baseada no ILA.

4.1.3 Ambiente de Ensino Teórico

O AIIP oferece um ambiente para o ensino teórico (Figura 4.4) na forma semelhante à de um livro clássico de programação, onde são disponibilizados conceitos e exemplos sobre programação estruturada em um contexto introdutório, divididos em unidades de conhecimento.

Figura 4.4: Tela do Ambiente Teórico do AIIP.



Fonte: Próprio autor.

O AIIP incentiva uma troca constante entre as abordagens teórico e prática, uma vez que o aluno só avança para unidades posteriores se tiver resolvido, com bom aproveitamento, parte dos problemas pertencentes à unidade atual.

4.1.4 Ambiente de Ensino Prático

Com o objetivo de desenvolver as soluções algorítmicas para solucionar os problemas armazenados na base de dados, o aluno tem à sua disposição o ambiente prático de ensino. Este ambiente oferece um editor de código com uma interface amigável para que o aluno possa desenvolver seus algoritmos. O editor possui algumas funcionalidades que ajudam o aluno na escrita dos seus algoritmos como, por exemplo: exibição do número de linhas de código e destaque das palavras reservadas da linguagem, facilitando a visualização e o entendimento do aluno no que se refere à construção do código.

É possível observar na Figura 4.5 a estruturação das funcionalidades deste ambiente prático. A parte identificada pelo número 1 representa a barra de ferramentas, a parte identificada pelo número 2 representa o editor de código, a parte identificada pelo número 3 representa o *display* de saída e a parte identificada pelo número 4 representa o simulador de código passo-a-passo.

Figura 4.5: Tela do Ambiente Prático do AIIP.

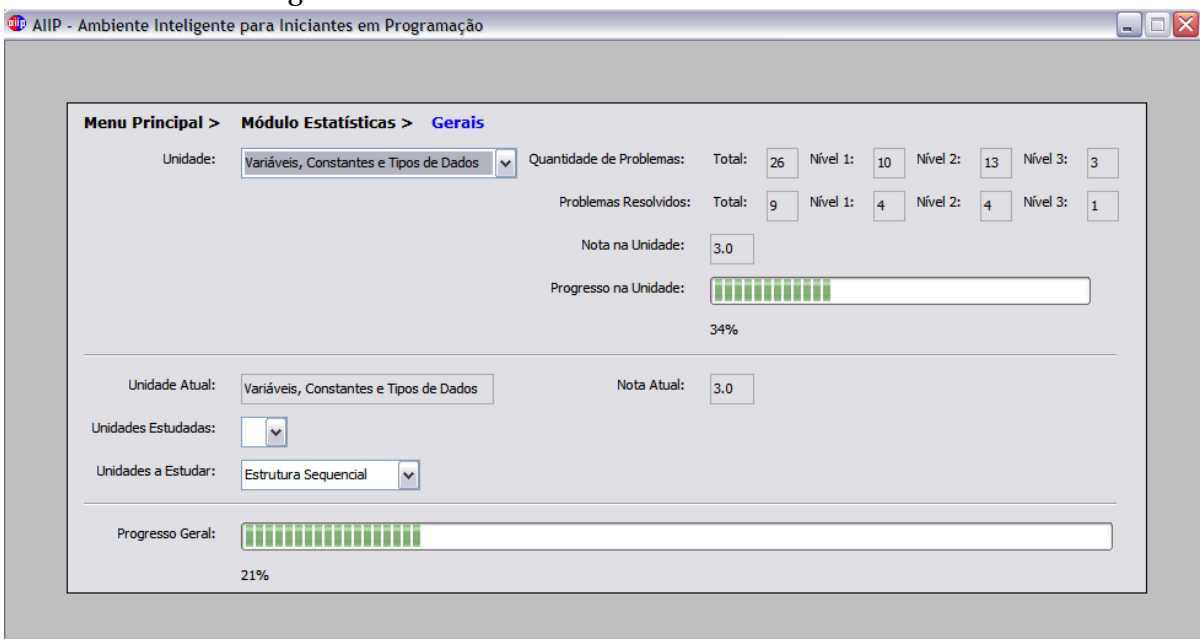
Fonte: Próprio autor.

4.1.5 Ambiente de Estatísticas

O Ambiente de Estatísticas (Figura 4.6) possibilita aos alunos e professores o acesso às estatísticas sobre os níveis de aprendizado dos alunos, além de informações gerais ou específicas sobre cada problema. Por exemplo: o professor pode ter acesso à quantidade de problemas resolvidos por um aluno em uma determinada unidade ou verificar quais foram

seus erros cometidos em problemas específicos e então tomar alguma atitude para auxiliar este aluno.

Figura 4.6: Tela do Ambiente de Estatísticas do AIIP.



Fonte: Próprio autor.

4.1.6 Ambiente de Gerenciamento

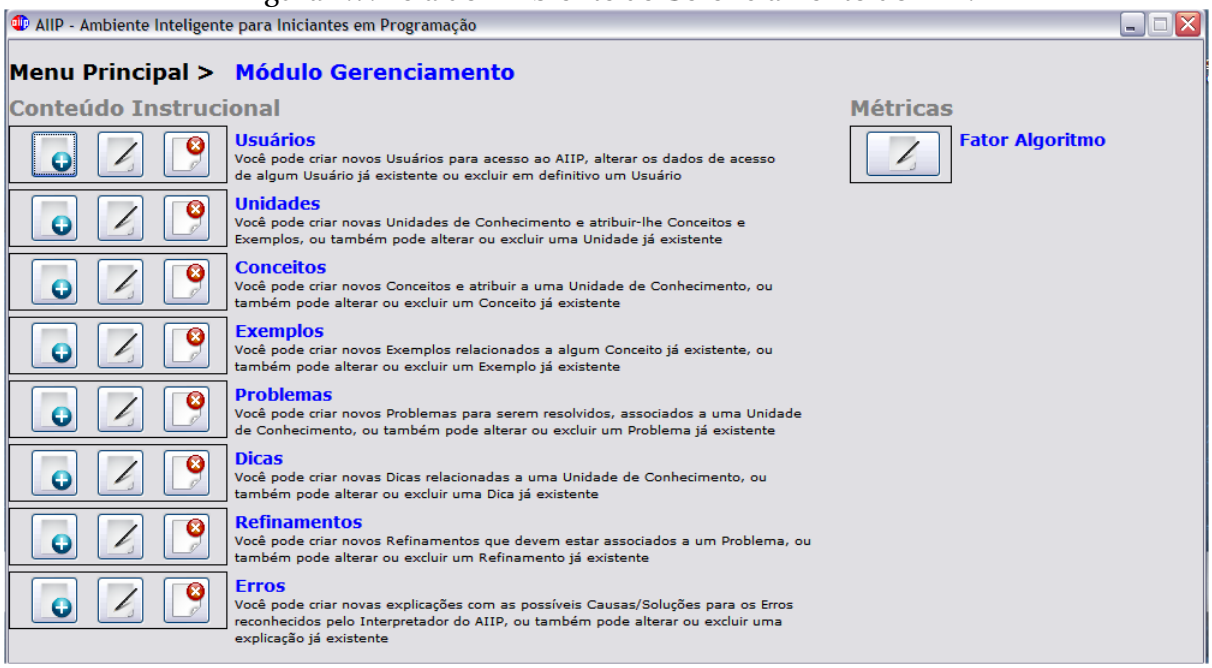
O Ambiente de Gerenciamento (Figura 4.7) possibilita ao professor o gerenciamento de todo o conteúdo instrucional (Unidades, Conceitos, Exemplos, Problemas, Dicas, Refinamentos e *Feedbacks*) utilizado pelo AIIP, além do gerenciamento dos usuários (Alunos e Professores) e das métricas utilizadas na avaliação automática das soluções algorítmicas.

É possível observar na Figura 4.8 um exemplo que simula o cadastramento na base de dados do AIIP de um problema através deste ambiente.

4.1.7 Assistente Inteligente

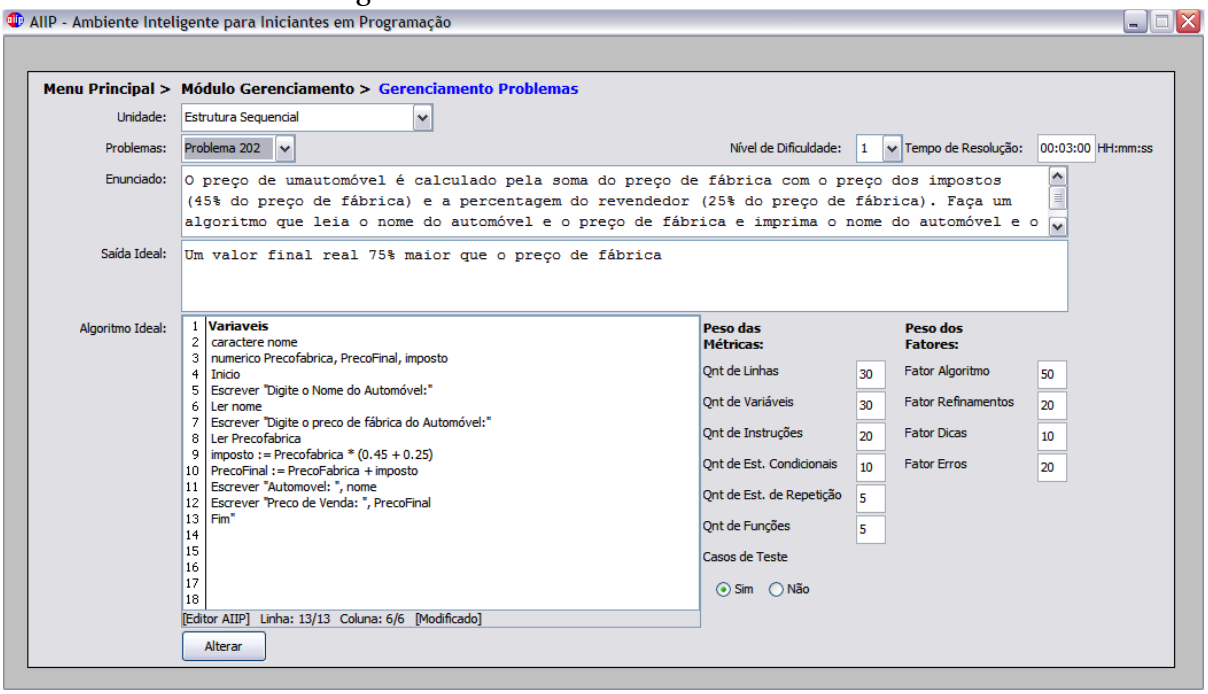
O assistente inteligente é o principal componente do AIIP e responsável principalmente por auxiliar os alunos na resolução dos problemas armazenados, fornecendo dicas e *feedbacks* apropriados, além de avaliar de forma automática as soluções algorítmicas dos alunos de acordo com métricas pré-estabelecidas. O capítulo 6 apresenta o assistente inteligente juntamente com seus objetivos e funções de forma mais detalhada.

Figura 4.7: Tela do Ambiente de Gerenciamento do AIIP.



Fonte: Próprio autor.

Figura 4.8: Gerenciamento de Problemas.



Fonte: Próprio autor.

4.1.8 Avaliação Automática

O componente de Avaliação Automática é responsável pela organização das pontuações e pesos das métricas dos problemas, utilizados pelo assistente inteligente para realizar as avaliações automáticas das soluções algorítmicas dos alunos.

4.1.9 Recomendação de Conteúdo

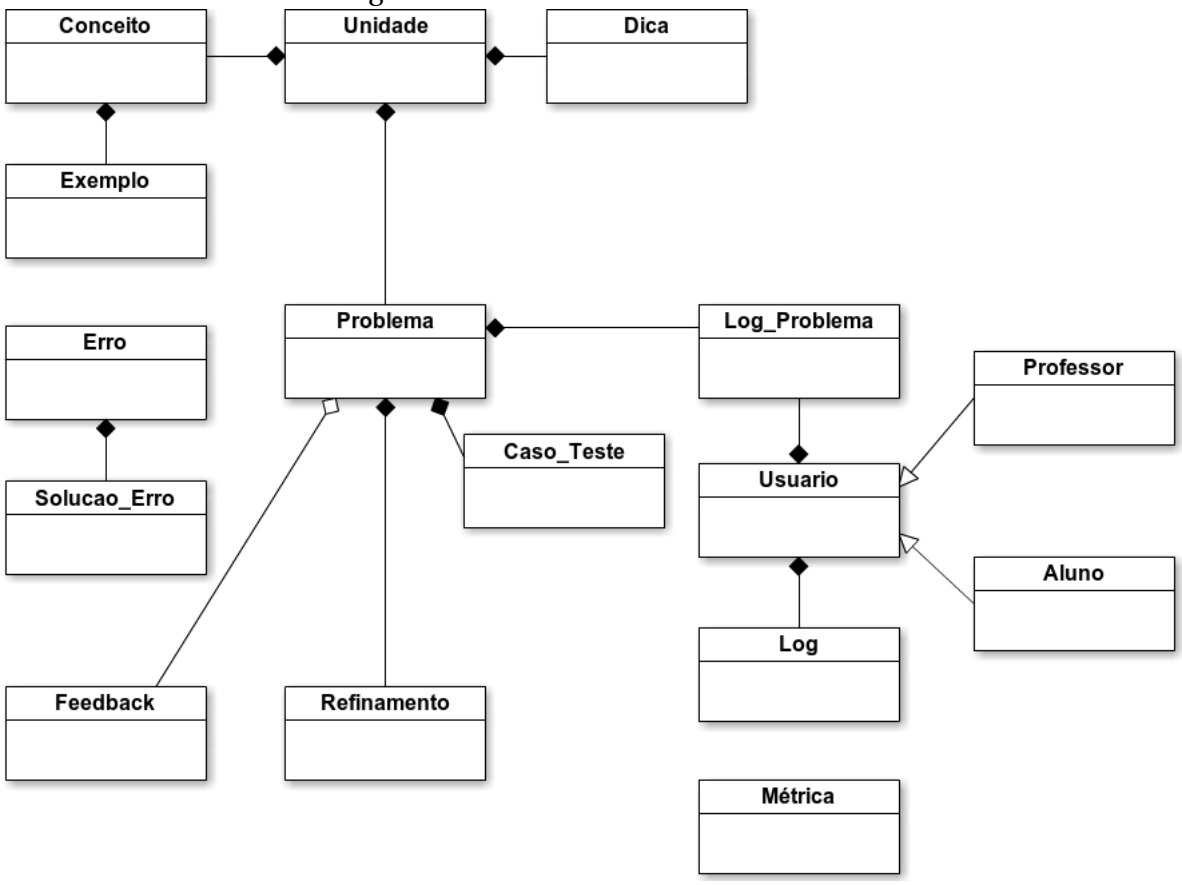
O componente de Recomendação de Conteúdo é responsável por indicar ao aluno, através do assistente inteligente, as unidades, conceitos ou problemas adequados de acordo com o seu nível de conhecimento atual, no momento da interação com o AIIP. Por exemplo, quando um aluno avança de unidade novos problemas são liberados para que ele tente resolvê-los. No entanto, a depender do seu nível de conhecimento atual, alguns problemas ficam inacessíveis, esperando que o aluno alcance um nível de conhecimento compatível com estes problemas.

4.1.10 Base de Dados

É na Base de Dados que fica armazenado todo o banco de dados do AIIP, acessado pelo assistente inteligente. Na Figura 4.9 é representado o modelo conceitual do AIIP, onde são apresentados os principais elementos existentes no ambiente.

As descrições destes elementos, representados através de classes e seus relacionamentos, podem ser observadas de forma mais detalhada no Apêndice C.

Figura 4.9: Modelo conceitual do AIIP.



Fonte: Próprio autor.

5 O ASSISTENTE INTELIGENTE DO AIIP

Sabendo das dificuldades existentes para a construção de um STI completo, com todas as suas características e funcionalidades, principalmente no domínio da programação, optou-se pela implementação de um assistente inteligente, com o objetivo de auxiliar os alunos e professores no processo de ensino/aprendizagem de programação.

É importante deixar claro que o assistente inteligente do AIIP não se trata de um STI como os definidos por Wenger (1987), uma vez que não possui todas as características essenciais comuns a estes sistemas, como por exemplo, um módulo de estratégias pedagógicas bem definidas. A proposta apresentada aqui se trata de um assistente no sentido de ser um auxiliador para o aluno no seu processo de aprendizagem, além de auxiliar o professor no gerenciamento do material instrucional e na avaliação dos alunos.

Mais precisamente, o objetivo principal do assistente inteligente do AIIP é a observação das ações realizadas por um aluno durante o processo de construção da solução algorítmica, com objetivo de resolver algum problema proposto, e sua reação da maneira mais apropriada àquele aluno, naquele determinado momento. Com relação aos professores, o objetivo principal do assistente inteligente do AIIP é o auxílio no gerenciamento do conteúdo instrucional bem como na avaliação das soluções algorítmicas dos alunos de forma automática de acordo com métricas pré-estabelecidas. As principais funções do assistente inteligente do AIIP são:

- Fornecimento de dicas e refinamentos aos alunos durante a resolução dos problemas armazenados;
- Identificação de erros cometidos pelos alunos durante a resolução de um problema específico e apresentação das possíveis causas e soluções para estes erros;
- Fornecimento de *feedbacks* apropriados aos alunos, levando em consideração os seus perfis;
- Realização dos cálculos necessários para a avaliação automática das soluções algorítmicas dos alunos de acordo com métricas pré-estabelecidas;
- Auxílio ao professor no gerenciamento do conteúdo instrucional;
- Auxílio ao professor no gerenciamento das métricas para avaliação das soluções algorítmicas dos alunos.

5.1 Ativação do Assistente Inteligente do AIIP

Quando o aluno está trabalhando no ambiente teórico do AIIP, o assistente inteligente não reage às suas ações, no sentido de interromper o aluno ou apresentar alguma dica,

refinamento ou *feedback*, ou seja, o aluno pode navegar entre as unidades de conhecimento e os conceitos sem que o assistente seja ativado.

Uma vez que o aluno comece a utilizar o ambiente prático do AIIP, com o objetivo de resolver um problema específico armazenado na ambiente, o assistente inteligente é ativado e começa a trabalhar em segundo plano, carregando informações sobre o aluno e sobre o problema em questão.

A ativação do assistente inteligente, e sua possível interferência junto ao aluno, é realizada automaticamente nas seguintes situações:

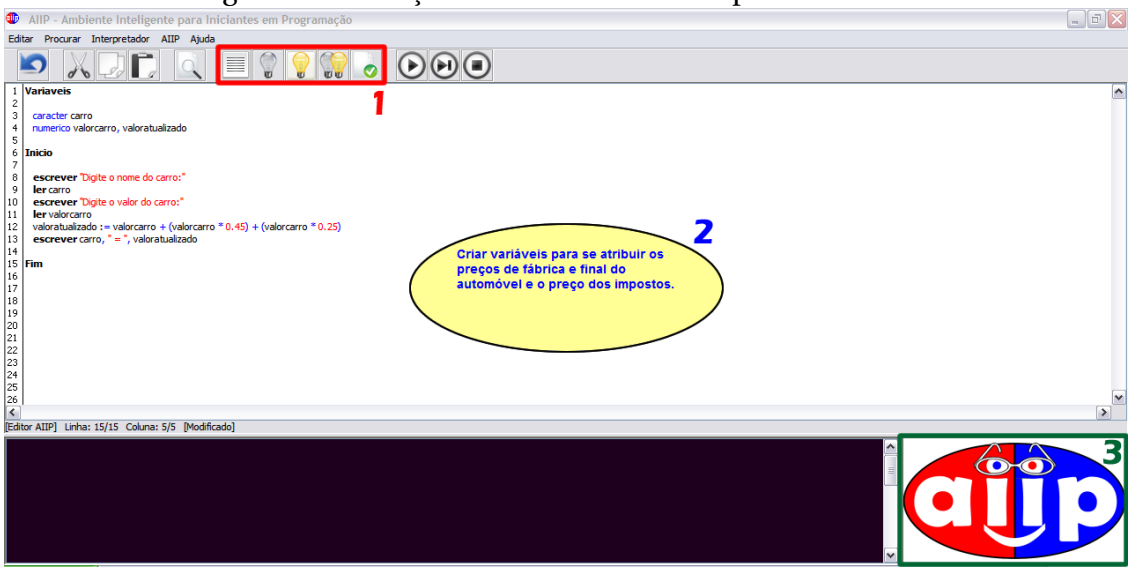
- Após o início da resolução de um problema pelo aluno;
- Após cada ação do aluno de salvar seu algoritmo;
- Após cada ação do aluno de solicitar uma dica;
- Após cada ação do aluno de solicitar um refinamento;
- Após o aluno submeter sua solução para ser avaliada.

Todas as informações geradas no processo de resolução do problema, inclusive a ocorrência de erros, são armazenadas para serem tratadas posteriormente, caso seja necessário.

O assistente é ativado quando o aluno resolve salvar o seu algoritmo, verificando a solução do aluno em busca de algum erro e, caso encontre, apresentando um *feedback* específico ao aluno.

Ao solicitar uma dica ou um refinamento, o assistente também é ativado, verificando informações sobre o aluno e sobre o problema, e fornecendo, ou não, os recursos solicitados.

Figura 5.1: Exibição de um Refinamento pelo assistente.



Fonte: Próprio autor.

Por exemplo, pode-se observar na Figura 5.1 a exibição de um refinamento (2) pelo assistente (3) através da solicitação realizada pelo menu exclusivo do assistente (1). Algumas métricas, explicadas posteriormente, são levadas em conta pelo assistente para o fornecimento destes recursos.

Além das formas de ativação anteriores, o assistente inteligente também é ativado quando o aluno submete sua solução para ser avaliada, realizando cálculos a partir de métricas pré-estabelecidas para atribuir uma nota final à solução algorítmica do aluno com relação ao problema específico ao qual o aluno estava resolvendo.

5.2 Fornecendo Dicas e Refinamentos

Objetivando oferecer recursos pedagógicos adicionais à nossa proposta, levando em consideração as características listadas por Muñoz-Merino & Kloos (2009), foi utilizada uma abordagem metodológica de incentivo aos alunos na busca da solução para os problemas propostos, não fornecendo explicitamente a resposta aos mesmos, mas através do fornecimento de dois tipos de dicas que podem ser apresentadas aos alunos durante o processo de resolução dos problemas: as **Dicas Gerais** e as **Dicas Específicas do Problema**, que neste caso optou-se por chamar de **Refinamentos** da solução.

5.2.1 Dicas Gerais

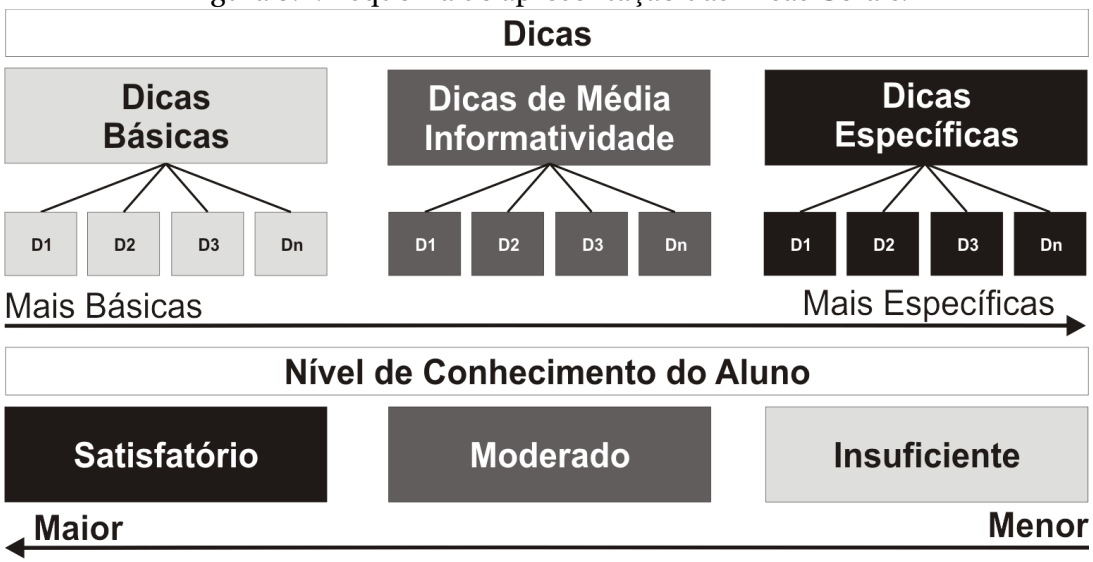
As Dicas Gerais armazenadas no AIIP possuem um esquema hierárquico de apresentação e estão relacionadas ao conteúdo programático da unidade de conhecimento a qual pertence o problema que o aluno encontra-se tentando resolver no momento em que solicita uma dica deste tipo. Estas dicas são categorizadas em três níveis: N = 1 (**Dicas Básicas**), N = 2 (**Dicas de Média Informatividade**), N = 3 (**Dicas Específicas**).

Esta divisão das Dicas Gerais em níveis é necessária uma vez que para a apresentação destas dicas o assistente inteligente do AIIP analisa o nível de conhecimento do aluno para verificar a qual destes níveis de dicas o mesmo está apto a solicitar e receber auxílio.

Sendo assim, também foi necessário categorizar o nível do conhecimento do aluno com relação ao domínio. Assim como as Dicas Gerais, o nível de conhecimento do aluno sobre o domínio foi divido em 3 níveis: **Conhecimento Insuficiente**, **Conhecimento Moderado**, **Conhecimento Satisfatório**.

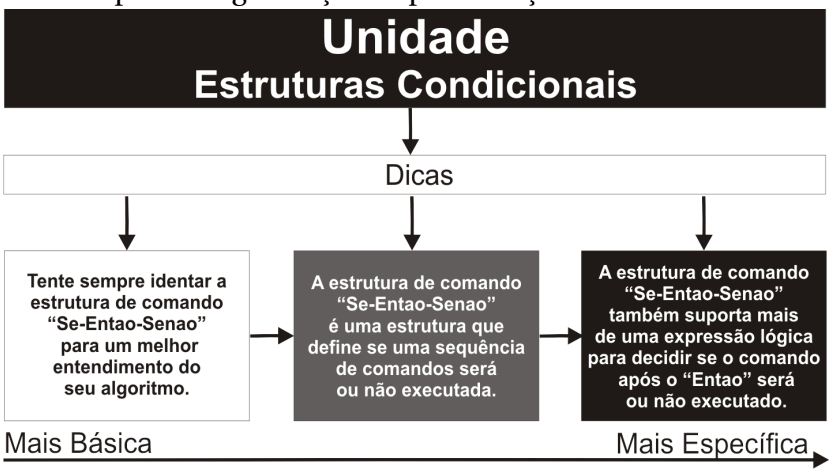
Observa-se na Figura 5.2 o esquema de apresentação destas dicas, partindo das dicas mais básicas até as dicas mais específicas pertencentes a uma unidade, levando em consideração o nível de conhecimento do aluno. Já na Figura 5.3, é apresentado um exemplo de organização e consequente apresentação das dicas pertencentes a uma unidade de conhecimento que aborda conceitos relacionados às estruturas condicionais.

Figura 5.2: Esquema de apresentação das Dicas Gerais.



Fonte: Próprio autor.

Figura 5.3: Exemplo de organização e apresentação das Dicas de uma Unidade.



Fonte: Próprio autor.

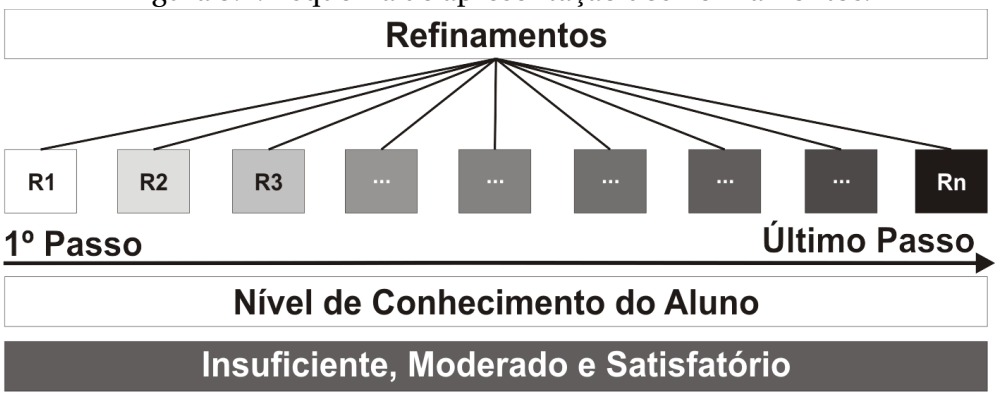
5.2.2 Refinamentos

As Dicas Específicas do Problema foram denominadas Refinamentos uma vez que as mesmas, a cada vez que são apresentadas, vão refinando a solução do problema para o aluno, auxiliando-o desta forma no processo de resolução.

Cada problema possui um número específico de refinamentos, que estarão disponíveis para apresentação após a solicitação dos alunos.

Diferentemente das Dicas Gerais os Refinamentos não são divididos em níveis, uma vez que cada refinamento sempre representará uma etapa ou passo posterior, em relação ao seu refinamento antecessor, para que o aluno chegue à resolução do problema em que estiver trabalhando, conforme observa-se na Figura 5.4.

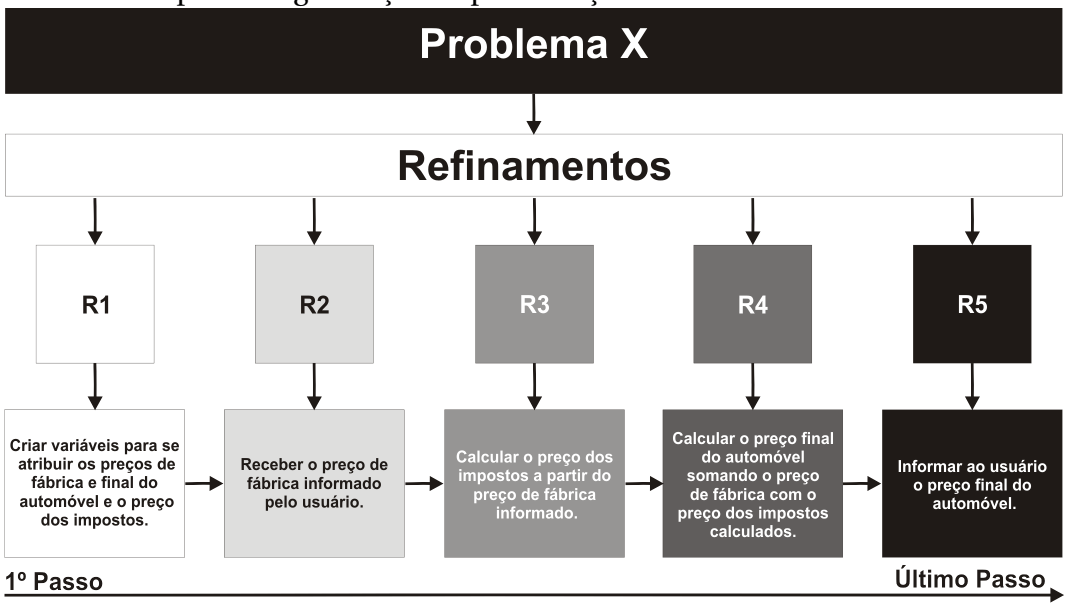
Figura 5.4: Esquema de apresentação dos Refinamentos.



Fonte: Próprio autor.

Na Figura 5.5 observa-se um exemplo de como os refinamentos são organizados, possuindo uma ordem de apresentação com relação a um problema específico. Para exemplificar foi utilizado um problema simples, cujo enunciado é: *"O preço de um automóvel é calculado pela soma do preço de fábrica com o preço dos impostos (45% do preço de fábrica) e a percentagem do revendedor (25% do preço de fábrica). Faça um algoritmo que leia o nome do automóvel e o preço de fábrica e imprima o nome do automóvel e o preço final"*.

Figura 5.5: Exemplo de organização e apresentação dos Refinamentos de um Problema.



Fonte: Próprio autor.

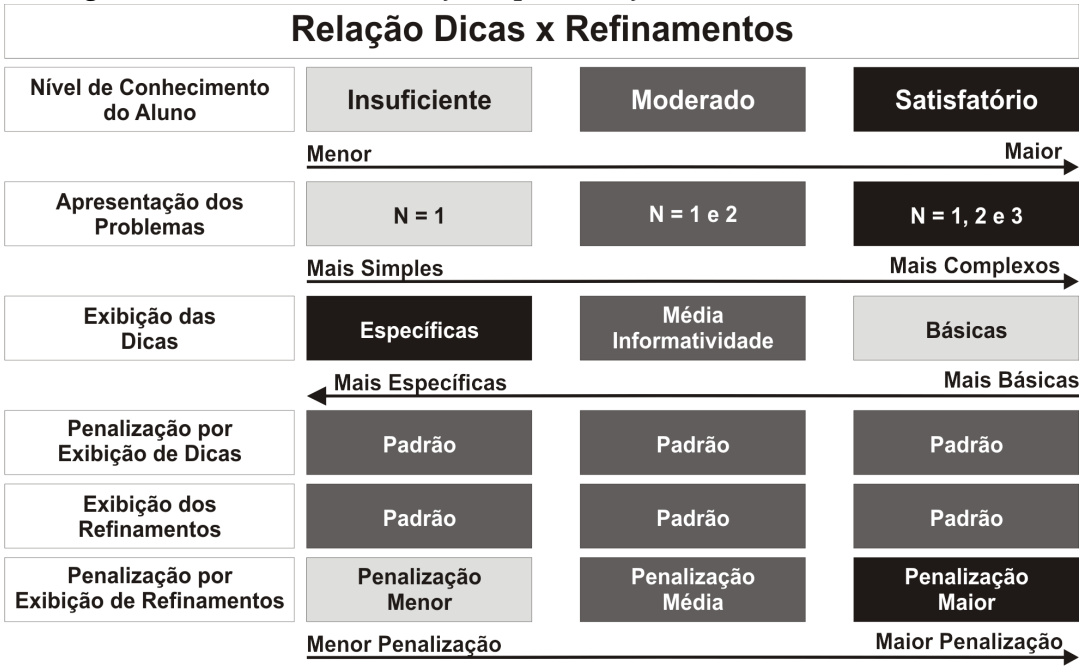
5.2.3 Comparativo Dicas x Refinamentos

Vale a pena ressaltar que, tanto a apresentação das Dicas, quanto a apresentação dos Refinamentos, possuem influência na formação da nota final do aluno, atribuída pelo assistente

inteligente, com relação à sua solução algorítmica em resposta ao problema em questão.

Desta forma, conforme pode-se observar na Figura 5.6, as Dicas são apresentadas ao aluno de acordo com o nível de conhecimento do mesmo em relação ao domínio. Quanto maior o nível de conhecimento do aluno, com relação ao domínio, mais básica é a dica apresentada a ele, consequentemente, quanto menor o nível de conhecimento do aluno, mais especifica é a dica apresentada ao mesmo para que ele consiga chegar a solução do problema que estiver tentando solucionar. Já a penalização pela apresentação destas dicas independe do nível de conhecimento do aluno, ou seja, o assistente inteligente leva em consideração apenas a quantidade de dicas apresentadas.

Figura 5.6: Estrutura de exibição e penalização das Dicas e Refinamentos.



Fonte: Próprio autor.

Com relação aos Refinamentos, não existe uma diferenciação na apresentação dos mesmos, levando em consideração o nível de conhecimento do aluno. Porém, a penalização pela apresentação de um Refinamento é maior para um aluno com nível satisfatório de conhecimento do que para um aluno com nível de conhecimento insuficiente.

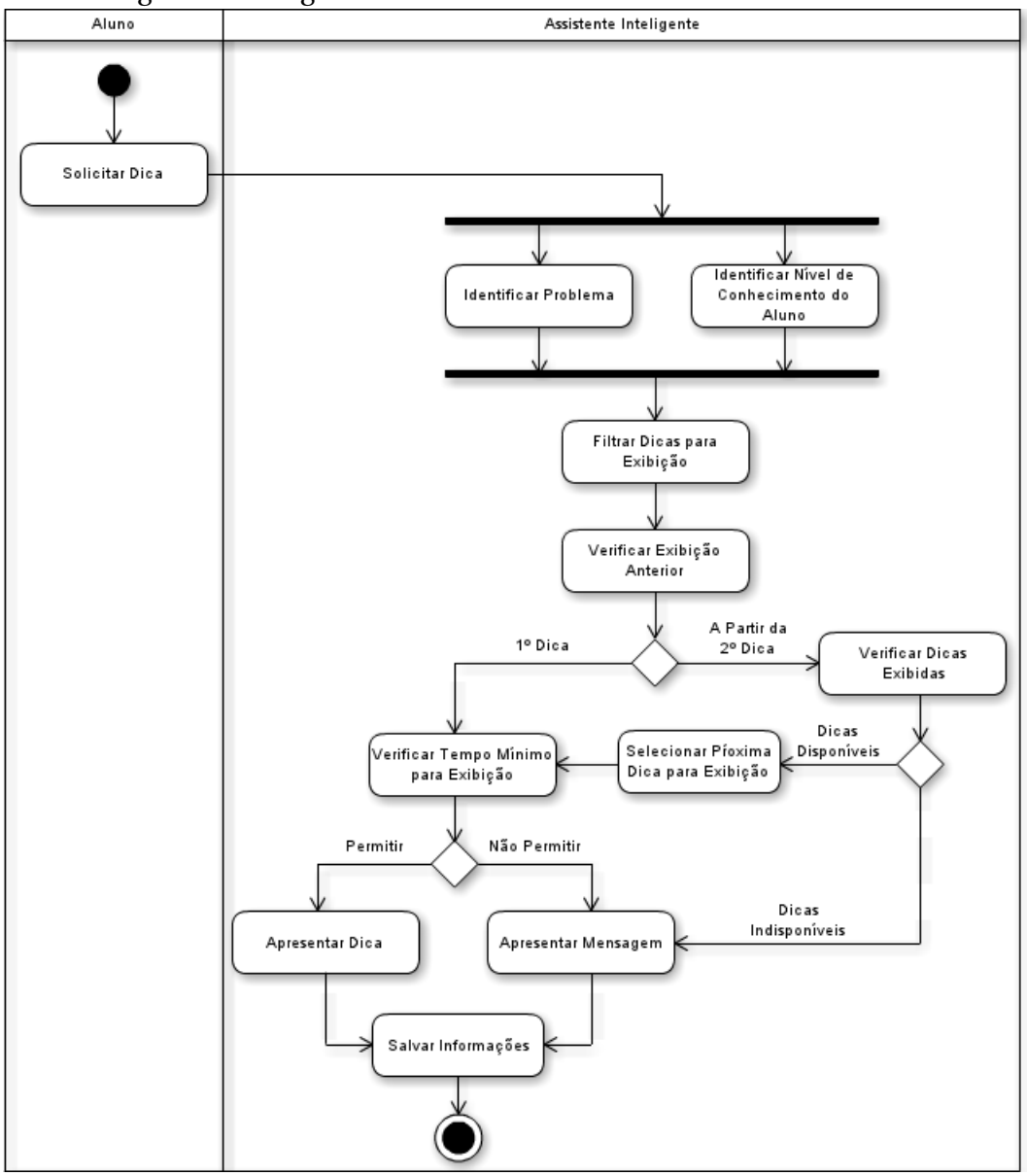
A princípio, a metodologia utilizada na apresentação das Dicas Gerais pode parecer um pouco contraditória. Porém, como o aluno ainda não possui conhecimento satisfatório sobre o domínio, acredita-se que o mesmo deve ser auxiliado com dicas mais específicas, que tentem suprir com mais precisão as suas dúvidas. Com relação aos alunos que possuem um conhecimento maior sobre o domínio, seria muito mais fácil se estes recebessem dicas mais especificas, uma vez que eles já possuem algum conhecimento sobre o domínio. Neste sentido, as dicas mais básicas são oferecidas para estes alunos.

Ou seja, não seria justo penalizar um aluno com um nível de conhecimento insuficiente

da mesma forma como um aluno com um nível de conhecimento satisfatório, se estes recebessem dicas mais básicas e mais especificas, respectivamente.

O processo para apresentação das Dicas e dos Refinamentos é bem parecido. Na Figura 5.7 observa-se o diagrama de atividades do processo de apresentação das Dicas Gerais aos alunos.

Figura 5.7: Diagrama de atividades do fornecimento das Dicas.



Fonte: Próprio autor.

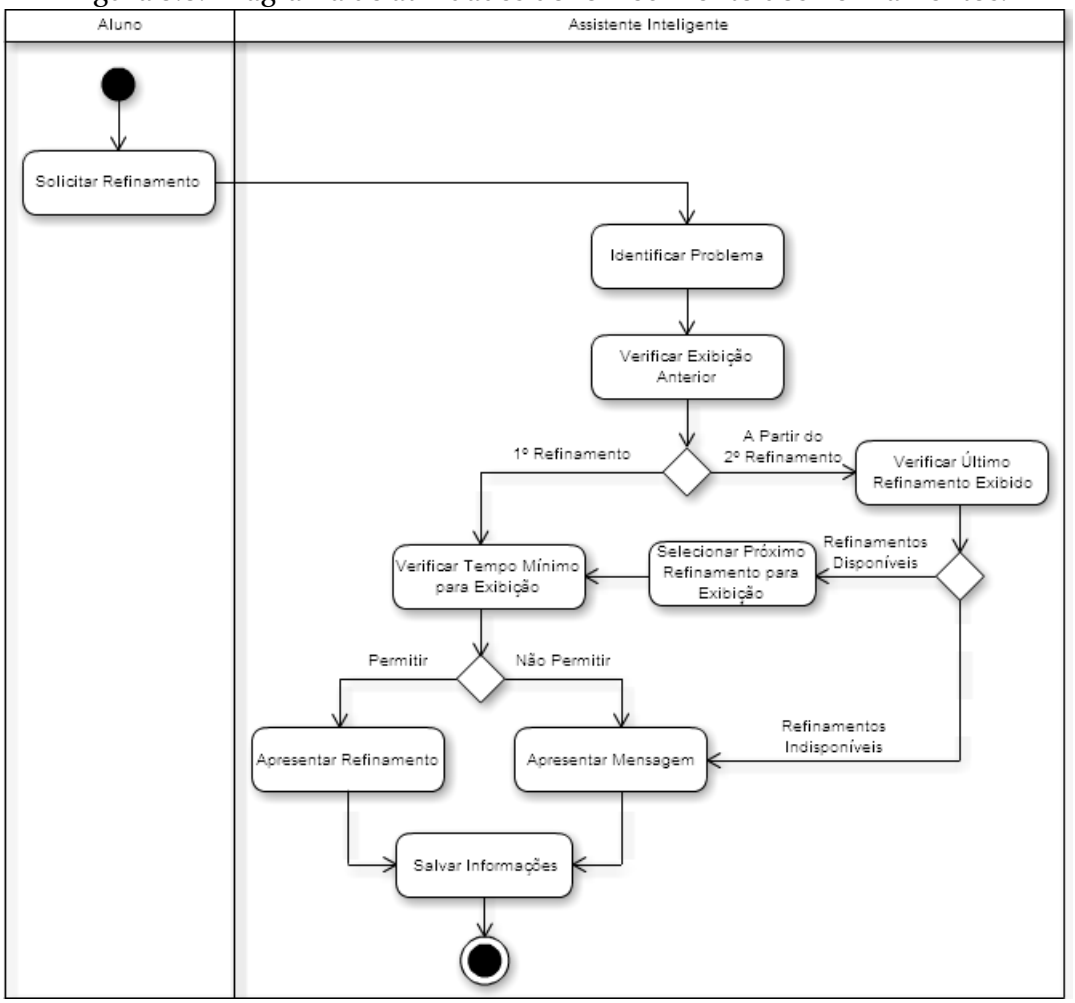
A primeira etapa é identificar o problema que o aluno está tentando resolver, verificando também o seu nível de dificuldade e a qual unidade de conhecimento o mesmo está associado, além de identificar o nível de conhecimento atual do aluno. A seguir, as dicas são filtradas e o assistente verifica, de acordo com a sessão atual, se alguma outra dica já foi exibida anteriormente para o problema em questão.

Caso a dica seja a primeira a ser exibida para o problema em questão, o assistente verifica se esta pode ser exibida, de acordo com o tempo mínimo necessário para a exibição da primeira dica. Se a dica puder ser exibida, o assistente exibe a dica e grava esta informação (solicitação e exibição de dica). Se a dica não puder ser exibida, o assistente exibe uma mensagem informando que a dica ainda não pode ser exibida e grava esta informação (solicitação e não exibição de dica).

Caso alguma dica já tenha sido exibida anteriormente para o problema atual, o assistente verifica quais foram as dicas já exibidas e prepara uma próxima dica para exibição, verificando se a dica pode ser exibida, de acordo com o tempo mínimo necessário para a exibição. Se a dica puder ser exibida, o assistente exibe a dica e grava esta informação (solicitação e exibição de dica). Se a dica não puder ser exibida, o assistente exibe uma mensagem informando que a dica ainda não pode ser exibida e grava esta informação (solicitação e não exibição de dica).

Caso a última dica, exibida anteriormente, seja também a última dica disponível para exibição, uma mensagem é exibida ao usuário informando que não há mais dicas disponíveis.

Figura 5.8: Diagrama de atividades do fornecimento dos Refinamentos.



Fonte: Próprio autor.

Para a apresentação dos Refinamentos, observa-se na Figura 5.8, como se desenvolve este processo, através de seu diagrama de atividades.

A primeira etapa é identificar o problema que o aluno está tentando resolver, selecionando os refinamentos relacionados à este problema. A seguir o assistente inteligente verifica, de acordo com a sessão atual, se algum outro refinamento já foi exibido anteriormente para o problema em questão.

Caso o refinamento seja o primeiro a ser exibido para o problema em questão, o assistente verifica se este pode ser exibido, de acordo com o tempo mínimo necessário para a exibição do primeiro refinamento. Se o refinamento puder ser exibido, o assistente exibe o refinamento e grava esta informação (solicitação e exibição de refinamento). Se o refinamento não puder ser exibido, o assistente exibe uma mensagem informando que o refinamento ainda não pode ser exibido e grava esta informação (solicitação e não exibição de refinamento).

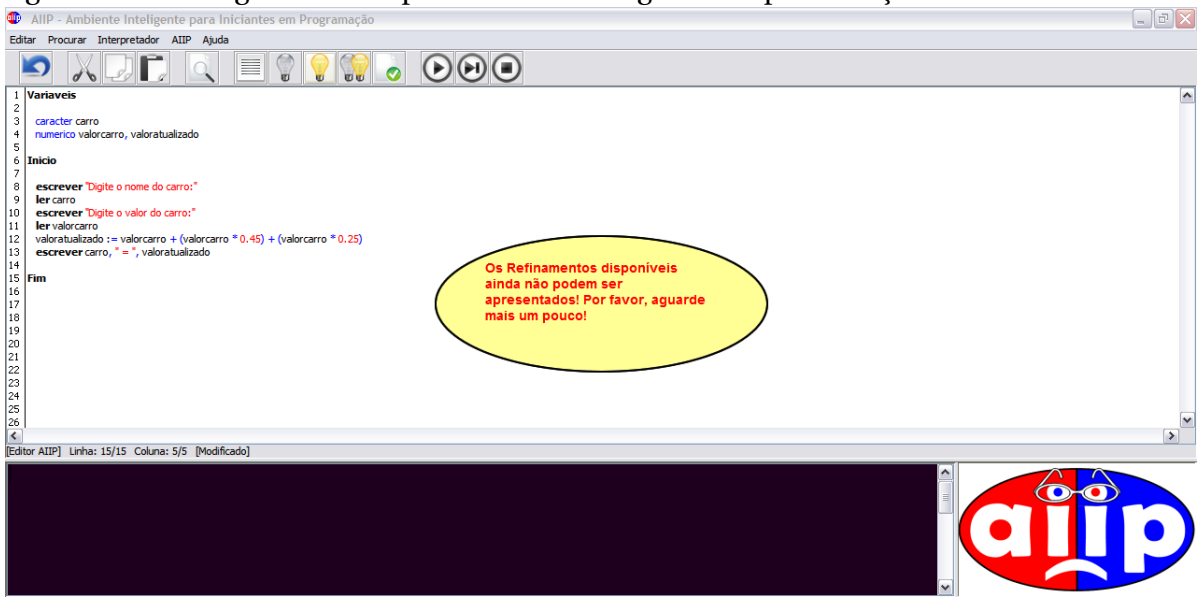
Caso algum refinamento já tenha sido exibido anteriormente para o problema atual, o assistente inteligente verifica qual foi o último refinamento exibido e prepara o próximo refinamento para exibição, verificando se o refinamento pode ser exibido, de acordo com o tempo mínimo necessário para a exibição. Se o refinamento puder ser exibido, o assistente exibe o refinamento e grava esta informação (solicitação e exibição de refinamento). Se o refinamento não puder ser exibido, o assistente exibe uma mensagem informando que o refinamento ainda não pode ser exibido e grava esta informação (solicitação e não exibição de refinamento).

Caso o último refinamento, exibido anteriormente, seja também o último refinamento do problema, uma mensagem é exibida ao usuário informando que não há mais refinamentos disponíveis.

É importante perceber que, para ambos os tipos de dicas do AIIP, **Dicas Gerais** e **Refinamentos**, foram elaboradas mensagens a serem exibidas aos alunos para evitar o "abuso de ajuda" destes recursos, conforme pode-se observar na Figura 5.9.

Estas mensagens levam em consideração a disponibilidade das dicas e refinamentos para exibição e o tempo mínimo para a exibição de cada um destes recursos. Por exemplo, se o aluno recebeu o número máximo de dicas ou refinamentos disponíveis para exibição, uma mensagem do tipo: *"Infelizmente não há mais dicas para exibir"* é apresentada ao aluno. Já se o aluno solicita de forma constante uma dica ou refinamento em um curto período de tempo, o assistente verifica se a dica ou o refinamento podem, ou não, ser exibidos. Se estes recursos ainda não estiverem disponíveis, uma mensagem do tipo: *"As dicas disponíveis ainda não podem ser apresentadas. Por favor, aguarde mais um pouco!"* é apresentada ao aluno.

Figura 5.9: Mensagem exibida pelo assistente negando a apresentação de um Refinamento.



Fonte: Próprio autor.

5.3 Fornecendo *Feedbacks*

Além do fornecimento de dicas, muito ambientes também fornecem *feedbacks* ao aluno, permitindo-o avaliar sua evolução no processo de aprendizagem. Tais *feedbacks* podem ser divididos em níveis, sendo mais simples ou mais detalhados, ou ainda divididos em positivos, transmitidos ao aluno para reforçar, elogiar ou parabenizar uma resposta correta, ou negativos, usados para corrigir uma resposta errada, justificando sobre o motivo da resposta estar errada, por exemplo.

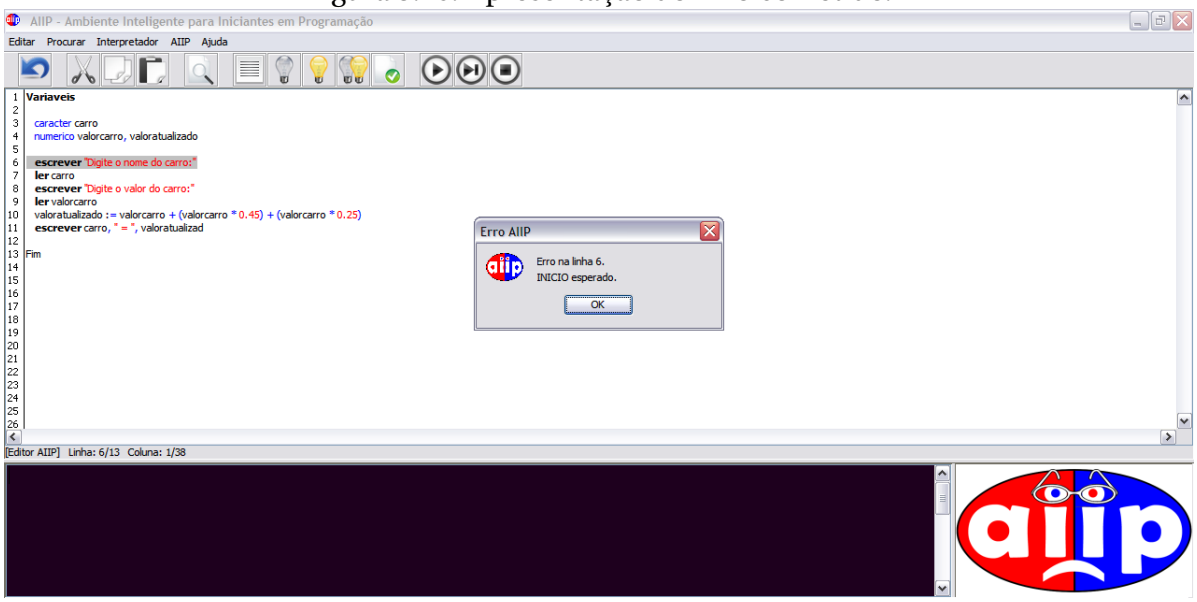
Um *feedback* obtido imediatamente após um erro cometido, após a identificação que o aluno está tendo dificuldades, ou mesmo após a avaliação da sua solução pode ajudar o aluno a eliminar equívocos na sua aprendizagem. Em muitos ambientes, a análise para o fornecimento de *feedbacks* leva em consideração apenas os erros cometidos pelos alunos durante a construção da solução, enquanto em outros a coerência da codificação com a sintaxe da linguagem também é levada em conta.

A preocupação com a forma de como se fornecer *feedbacks* vem sendo motivo de estudo pelos especialistas nos últimos anos. No trabalho de Mayer et al. (2006), por exemplo, os autores realizaram um experimento e identificaram que os alunos, principalmente aqueles com menos experiência no domínio tratado, reagem melhor com relação ao nível de polidez do *feedback* recebido. Por exemplo, *feedbacks* do tipo: *"Você pode pressionar a tecla Enter"* ou *"Vamos clicar na tecla Enter"* parecem ser melhor recebidos e aceitos do que *feedbacks* do tipo: *"Pressione a tecla Enter agora"* ou *"O sistema exige que você clique na tecla Enter"*.

É possível perceber a que níveis de detalhes estas pesquisas sobre o fornecimento de *feedbacks* podem chegar. No entanto, apesar de entender a importância destas pesquisas,

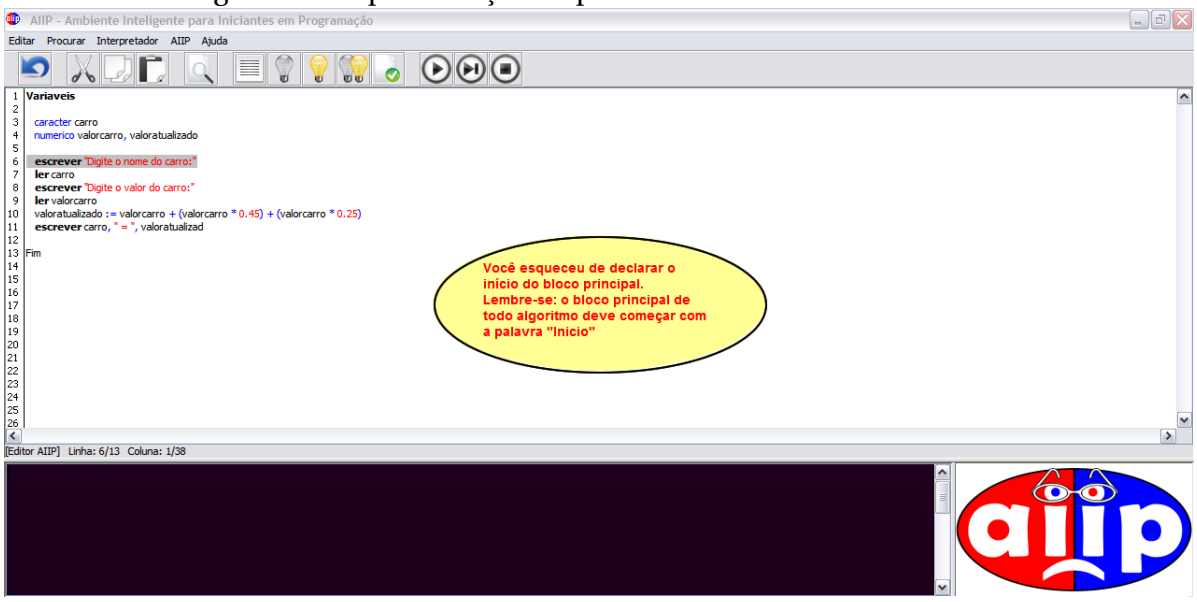
nosso objetivo principal não diz respeito à identificação de fatores relacionados à satisfação do aluno ou às suas reações aos *feedbacks* recebidos. Porém, é importante deixar claro que tentou-se elaborar os *feedbacks* a serem oferecidos pelo AIIP da forma mais polida possível, uma vez que a maioria dos alunos que utilizarão o ambiente serão iniciantes em programação.

Figura 5.10: Apresentação do Erro cometido.



Fonte: Próprio autor.

Figura 5.11: Apresentação da possível causa do Erro cometido.



Fonte: Próprio autor.

Os *feedbacks* foram divididos em 2 tipos: *feedbacks* de erros e *feedbacks* de problemas. Os *feedbacks* de erros estão relacionados aos erros reconhecidos pelo AIIP e são ap-

resentados toda vez que o aluno comete algum erro, informando ao aluno qual foi o erro cometido (Figura 5.10), as possíveis causas e também as possíveis soluções para corrigir o erro (Figura 5.11). Os *feedbacks* de problemas estão relacionados aos problemas armazenados no AIIP e são apresentados após a avaliação automática da solução algorítmica do aluno, de acordo com o resultado obtido na avaliação.

5.4 Avaliando Automaticamente as Soluções Algorítmicas

Durante o processo de desenvolvimento de um algoritmo, em resposta a algum problema proposto, os alunos podem se deparar com diversas situações que venham a facilitar ou dificultar sua capacidade de resolver tal problema. Por exemplo, se o aluno já possuir uma base de conhecimento matemática boa, resolver um problema que necessite de conhecimentos matemáticos pode ser uma tarefa mais fácil, diferentemente de uma situação onde o aluno não possui uma boa base matemática.

No entanto, se o aluno recebe alguma dica que lhe ajude, explicando-lhe por onde começar ou quais caminhos seguir para alcançar a solução do problema, é possível que este processo seja realizado com mais facilidade. Os erros cometidos também podem influenciar neste processo, uma vez que se o aluno não consegue identificar um erro cometido, ou se consegue identificar, mas não consegue corrigi-lo, talvez possa desistir ou achar que não tem capacidade para resolver o problema em questão.

Sendo assim, é possível elencar alguns aspectos e características de aprendizagem que são comuns ao domínio da programação de computadores com relação aos alunos iniciantes. Com isso, a ideia de avaliar o algoritmo de forma automática, levando em consideração não só o algoritmo em si, mas o seu processo de construção e desenvolvimento, pode ser considerado como uma alternativa diferente e atraente neste domínio.

Umas das principais contribuições do nosso trabalho é a proposta alternativa para a avaliação do processo de construção de um algoritmo, onde através de um conjunto de métricas pré-estabelecidas, passíveis de flexibilização, as soluções algorítmicas passam a ser avaliadas de forma automática.

As ferramentas para avaliação automática geralmente são utilizadas para correção de questões objetivas, como, por exemplo, questões de múltipla escolha. Questões abertas, discursivas, que aceitam várias interpretações como resposta, são praticamente impossíveis de serem analisadas computacionalmente. Cada palavra escrita a mais, ou um sinônimo usado pelo aluno no lugar de uma palavra da solução armazenada são fatores que poderiam ser considerados como errados em uma avaliação feita por um computador, mesmo estando correto o raciocínio do aluno.

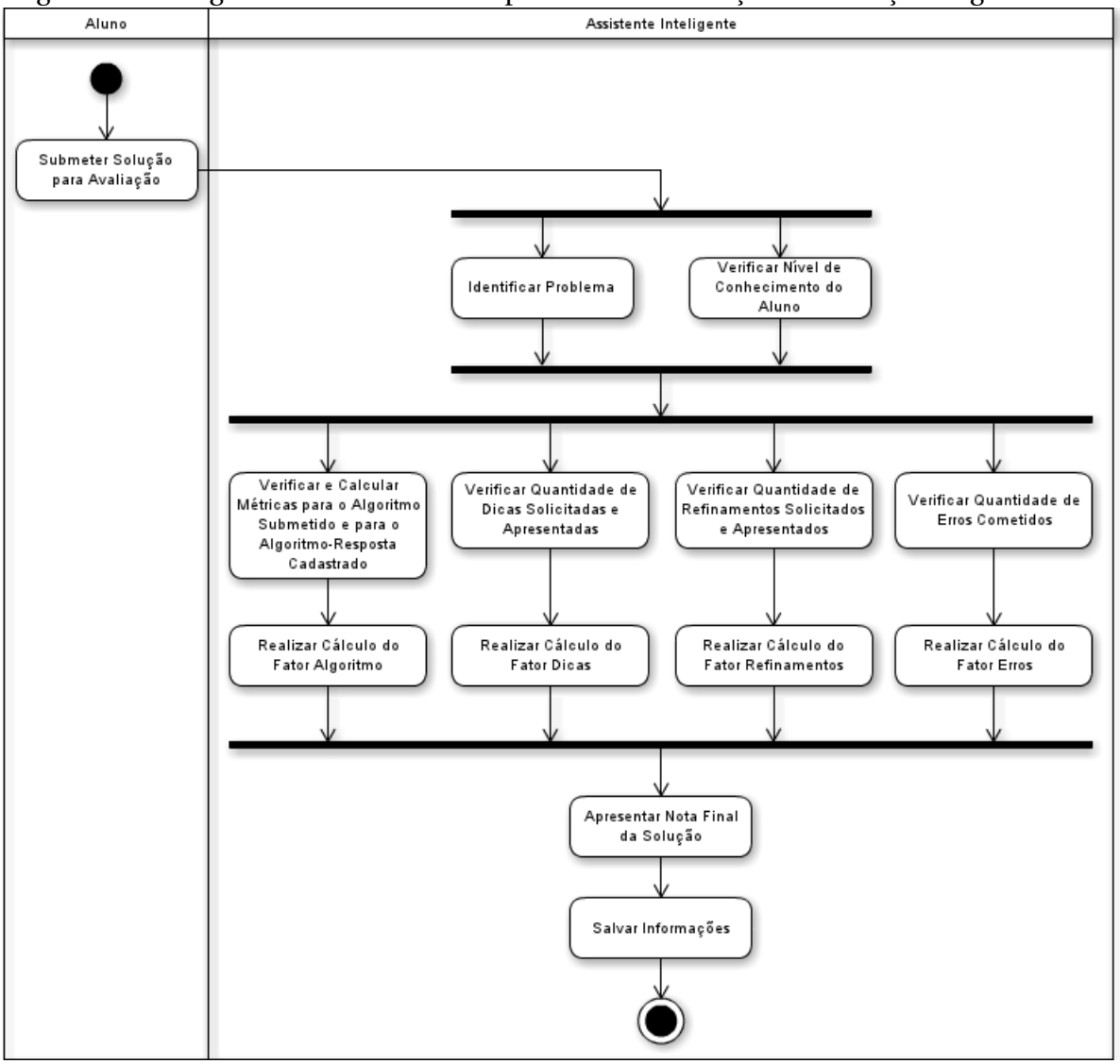
Desta forma, considerando um algoritmo como uma questão que possui resposta discursiva, passível de várias interpretações e várias soluções para um mesmo problema, entende-se

que sua avaliação de forma automática é um processo não necessariamente impossível, mas complexo.

Assim, para a avaliação e atribuição de uma nota para a solução algorítmica do aluno, este trabalho propõe uma análise separada de 4 fatores: o Algoritmo em si, as Dicas solicitadas, os Refinamentos solicitados e os Erros cometidos. Para cada um destes fatores propostos foram criadas métricas específicas com o objetivo de avaliar automaticamente as soluções algorítmicas, assim como, identificar o nível de conhecimento e as principais dificuldades dos alunos no domínio da programação.

Observa-se na Figura 5.12, de forma mais ampla, o processo de avaliação geral das soluções algorítmicas através de um diagrama de atividades. As métricas utilizadas e o detalhamento do processo de avaliação de cada um dos fatores propostos são apresentados nas subseções seguintes.

Figura 5.12: Diagrama de atividades do processo de avaliação das soluções algorítmicas.



Fonte: Próprio autor.

5.4.1 Métricas e Avaliação do Fator Algoritmo

Algumas das maneiras de se analisar algoritmos automaticamente se dá através da verificação de suas características ou através da comparação com outros algoritmos desenvolvidos para resolver um mesmo problema, sendo esta última forma o nosso principal foco.

Através da aplicação de um questionário (Apêndice A) a um grupo de professores da Universidade Federal de Alagoas que lecionam, ou já lecionaram, as disciplinas de Laboratório de Programação e/ou Introdução à Programação e/ou Programação 1, e após a consequente análise dos dados obtidos, foram desenvolvidas algumas métricas para analisar de forma automática as soluções algorítmicas desenvolvidas pelos alunos em resposta aos problemas apresentados aos mesmos no ambiente.

É importante destacar que as métricas desenvolvidas para a avaliação do fator Algoritmo prezam por uma análise quantitativa e não qualitativa do algoritmo, cabendo ao professor, ao final da avaliação, verificar se a nota atribuída de forma automática pelo assistente realmente condiz com a sua opinião.

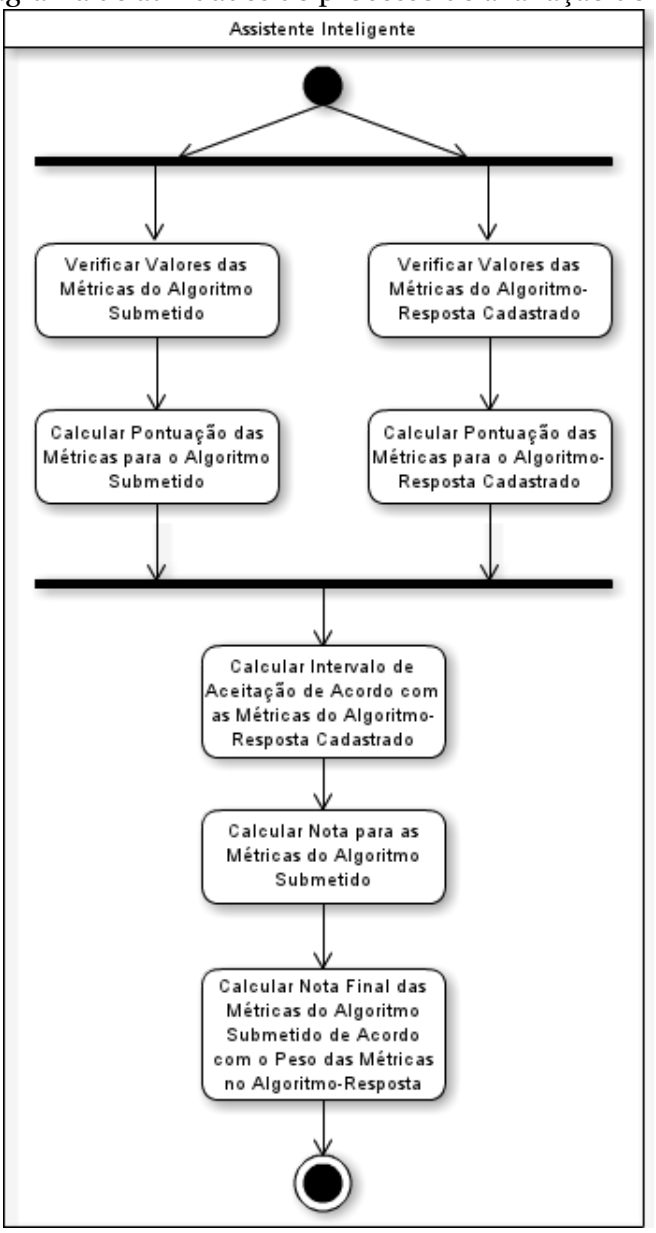
Em muitas ocasiões, alguns fatores importantes, como o tempo de execução do algoritmo e os recursos utilizados (memória), bem como a coerência das variáveis e funções em relação ao contexto teórico envolvido no problema são cruciais para validação e análise de um algoritmo. Porém, entendendo que a análise de forma automática destes e de outros fatores não citados tende a ser bastante complexa, a proposta apresentada neste trabalho parte para uma direção contrária, mais simples, onde analisa-se quantitativamente a ocorrência de certas estruturas comumente encontradas em algoritmos mais simples.

A metodologia para a avaliação automática dos algoritmos submetidos pelos alunos é bastante simples. Resumidamente, calculam-se os valores para cada métrica do algoritmo submetido e comparam-se com os valores armazenados de cada métrica do algoritmo-resposta cadastrado pelo professor, verificando ao final a equivalência entre os dois algoritmos. Na figura 5.13 pode-se observar o diagrama de atividades deste processo. Estas atividades serão explicadas de forma mais detalhada no decorrer desta seção.

Foram estabelecidas seis métricas que servirão para a análise dos algoritmos, são elas:

- ***quantidade de linhas de código***;
- ***quantidade de variáveis declaradas***;
- ***quantidade de instruções***;
- ***quantidade de estruturas condicionais***;
- ***quantidade de estruturas de repetição***;
- ***quantidade de funções***.

Figura 5.13: Diagrama de atividades do processo de avaliação do fator algoritmo.



Fonte: Próprio autor.

Para a métrica ***quantidade de linhas de código*** é considerado apenas o número de linhas utilizadas na solução algorítimica para a resolução do problema em questão, excluindo-se as linhas em branco e as linhas comentadas.

Para a métrica ***quantidade de variáveis declaradas*** é considerado apenas o número de variáveis declaradas na solução algorítmica para a resolução do problema em questão. Seria possível considerar outros fatores qualitativos, como por exemplo: as variáveis declaradas e não inicializadas ou a coerência dos nomes utilizados pelas variáveis com relação ao propósito ou ao tipo de dados aos quais elas se destinam a armazenar. No entanto, com o objetivo de facilitar o processo de avaliação automática, apenas o número total de variáveis declaradas é verificado.

Por exemplo, para os Algoritmos 5.1 e 5.2, o valor encontrado para a métrica ***quantidade de variáveis declaradas*** seria o mesmo: 7 (sete), variando apenas na quantidade de linhas de código utilizada e consequente organização visual do algoritmo.

Algoritmo 5.1 Exemplo 1.

```
Variaveis
    numerico a
    numerico b
    numerico c
    caractere nome
    caractere idade
    caractere endereco
    caractere profissao
Inicio
Fim
```

Algoritmo 5.2 Exemplo 2.

```
Variaveis
    numerico a, b, c
    caractere nome, idade, endereco, profissao
Inicio
Fim
```

Com relação à métrica ***quantidade de instruções*** é considerado apenas o número de instruções utilizadas na solução algorítmica para a resolução do problema em questão. São consideradas instruções:

- As expressões aritméticas (soma, subtração, multiplicação, divisão, potenciação);
- As expressões relacionais (maior que, menor que, igual, diferente, maior ou igual, menor ou igual);
- Os comandos de atribuição;
- Os comandos de entrada e saída (ler e escrever).

Com isso, é possível identificar se o algoritmo desenvolvido pelo aluno possui mais ou menos operações e comandos do que o necessário para se chegar à mesma solução, cabendo ao professor, ao final, penalizá-lo ou beneficiá-lo de acordo com a sua solução algorítmica.

O Algoritmo 5.3, por exemplo, possui 10 (dez) instruções, sendo 5 (cinco) comandos de entrada e saída, 3 (três) expressões e 2 (dois) comandos de atribuição.

Por fim, com relação às métricas ***quantidade de estruturas condicionais***, ***quantidade de estruturas de repetição*** e ***quantidade de funções*** é considerado apenas o número de ocorrências de cada uma destas estruturas encontradas no algoritmo. Para a métrica ***quantidade de estruturas condicionais***, verifica-se a quantidade de comandos *Se-Entao-Senao* e *Faca-Caso*,

Algoritmo 5.3 Exemplo 3.

```
Variaveis
    numerico a, b, c
Inicio
    Escrever "Digite o primeiro número"
    Ler a
    Escrever "Digite o segundo número"
    Ler b
    Se a > b Entao
        c := a + b
    Senao
        c := a * b
    Fim_se
    Escrever c
Fim
```

para a métrica ***quantidade de estruturas de repetição***, verifica-se a quantidade de comandos *Para-Proximo* e *Faca-Enquanto*, e para a métrica ***quantidade de funções*** verifica-se a quantidade de comandos *Funcao* existentes no algoritmo.

O Algoritmo 5.4 , por exemplo, possui 1 (uma) estrutura condicional, 2 (duas) estruturas de repetição e 2 (duas) funções.

Algoritmo 5.4 Exemplo 4.

```
Variaveis
    numerico a, b, c, x, y

Funcao soma(a, b)
Variaveis numerico c
Inicio
    c := a + b
    Retornar c
Fim

Funcao subtracao(a, b)
Variaveis numerico c
Inicio
    c := a - b
    Retornar c
Fim

Inicio
    Escrever "Digite o primeiro número"
    Ler a
    Escrever "Digite o segundo número"
    Ler b
    c := soma(a, b)
    x := 0
    y := 0
    Para x de 0 ate c Passo 1
        y := y + c
    Proximo
    Escrever y
    Se a > b Entao
        c := subtracao(a, b)
    Senao
        c := subtracao(b, a)
    Fim_se
    x := 0
    y := 0
    Para x de 0 ate c Passo 1
        y := y + c
    Proximo
    Escrever y
Fim
```

Na Tabela 5.1 estas métricas são apresentadas juntamente com uma pontuação correspondente à uma possível ocorrência de cada uma delas nos algoritmos submetidos pelos alunos. É importante deixar claro que estas pontuações são passíveis de flexibilização, ou seja, o professor tem autonomia para modificá-las de acordo com sua preferência. No entanto, a fim de explicar como os cálculos para a avaliação automática do fator Algoritmo são realizados, os valores contidos na Tabela 5.1 serão utilizados como exemplo deste ponto do trabalho em diante.

Tabela 5.1: Pontuação para a ocorrência de cada métrica.

Métrica	**Ponto por Métrica**
Quantidade de Linhas de Código	1
Quantidade de Variáveis Declaradas	2
Quantidade de Instruções	2
Quantidade de Estruturas Condicionais	3
Quantidade de Estruturas de Repetição	5
Quantidade de Funções	10

Fonte: Próprio autor.

Por exemplo, para cada linha de código, atribui-se 1 ponto. Desta forma, um algoritmo com 15 linhas receberá 15 pontos para a métrica ***quantidade de linhas de código***.

Como já explicado anteriormente, nosso objetivo é comparar o algoritmo submetido pelo aluno com o algoritmo-resposta cadastrado na base de dados do AIIP pelo professor, considerado ideal para a resolução do problema em questão.

Assim, em uma situação onde o problema que o aluno resolveu com 15 linhas (15 pontos), o seu algoritmo-resposta cadastrado pelo professor possua 12 linhas (12 pontos), a métrica ***quantidade de linhas de código*** receberia nota zero, pois a quantidade de linhas do algoritmo do aluno não é igual à quantidade de linhas do algoritmo-resposta cadastrado pelo professor.

Para evitar este tipo de problema, atribui-se a cada uma das métricas um intervalo de aceitação (Tabela 5.2).

Tabela 5.2: Intervalo de aceitação das métricas.

Métrica	**Intervalo de Aceitação**
Quantidade de Linhas de Código	20%
Quantidade de Variáveis Declaradas	30%
Quantidade de Instruções	30%
Quantidade de Estruturas Condicionais	20%
Quantidade de Estruturas de Repetição	30%
Quantidade de Funções	30%

Fonte: Próprio autor.

Este intervalo de aceitação representa um percentual aceitável (para mais e para menos) para a pontuação de cada métrica da solução do aluno, quando comparadas à pontuação de cada métrica do algoritmo-resposta cadastrado pelo professor. Assim, se a pontuação da métrica do algoritmo do aluno encontrar-se dentro do intervalo de aceitação com relação à pontuação da métrica do algoritmo-resposta cadastrado, então a métrica da solução do aluno também pode ser considerada como correta. Assim como a pontuação de cada métrica, estes intervalos também são passíveis de flexibilização, podendo o professor modificá-los de acordo com sua preferência. No entanto, os valores contidos na Tabela 5.2 serão utilizados como exemplo deste ponto do trabalho em diante.

Desta forma, com relação ao exemplo anterior, aplicando o índice de aceitação de 20%, ou seja, se a solução do aluno possuir entre 9 e 15 pontos (neste caso, entre 9 e 15 linhas), pode-se então considerar que o aluno obteve êxito nesta métrica, uma vez que os 12 pontos estão dentro do intervalo de aceitação.

Para cada uma das métricas analisada também atribui-se um peso relacionado à sua importância na formação da nota final do algoritmo do aluno. Estes pesos, assim como os outros dois itens (Pontuação e Intervalo de Aceitação), também são passíveis de flexibilização. No entanto, a fim de explicar os cálculos realizados, os pesos apresentados na Tabela 5.3 serão utilizados como exemplo deste ponto do trabalho em diante.

Tabela 5.3: Peso das métricas no código para a nota final.

Métrica	**Peso na Nota Final**
Quantidade de Linhas de Código	5%
Quantidade de Variáveis Declaradas	10%
Quantidade de Instruções	10%
Quantidade de Estruturas Condicionais	30%
Quantidade de Estruturas de Repetição	20%
Quantidade de Funções	25%
Total	**100%**

Fonte: Próprio autor.

Assim, seguindo ainda o exemplo anterior, observa-se que o peso da métrica ***quantidade de linhas de código*** na atribuição da nota para o algoritmo do aluno representa 5% (ou 0,5 ponto, considerando-se uma nota de 0 a 10). Desta forma, caso o assistente inteligente identifique que para esta métrica o aluno obteve êxito, então a nota máxima para esta métrica é atribuída, neste caso, 0,50 ponto. Caso o assistente identifique no algoritmo do aluno que a pontuação de alguma das métricas não está dentro do intervalo de aceitação, então um cálculo é realizado para a atribuição de uma nota equivalente para a métrica.

Caso a pontuação de uma métrica seja menor que a pontuação mínima aceitável, a nota para esta métrica deve ser calculada da seguinte forma: sua pontuação multiplicada por 10 e

dividida pela pontuação mínima aceitável para esta métrica no algoritmo-resposta, com o resultado desta operação multiplicado pelo peso da métrica, conforme pode-se observar na Equação 5.1, onde:

N: nota final da métrica.
P: pontuação da métrica obtida com relação ao código do aluno.
Pmin: pontuação mínima aceitável para esta métrica com relação ao algoritmo-resposta.
p: peso da métrica para a atribuição da nota final do algoritmo.

$$N = \left(\frac{10P}{Pmin}\right)p \tag{5.1}$$

Caso a pontuação da métrica seja maior que a pontuação máxima aceitável, a nota para esta métrica deve ser calculada da seguinte forma: sua pontuação multiplicada por 10 e dividida pela pontuação máxima aceitável para esta métrica no algoritmo-resposta, subtraindo de 20 o resultado encontrado e depois multiplicando o resultado pelo peso da métrica, conforme mostra a Equação 5.2, onde:

N: nota final da métrica.
P: pontuação da métrica obtida com relação ao algoritmo do aluno.
Pmax: pontuação máxima aceitável para esta métrica com relação ao algoritmo-resposta.
p: peso da métrica para a atribuição da nota final do algoritmo.

$$N = \left(20 - \frac{10P}{Pmax}\right)p \tag{5.2}$$

A subtração por 20 significa que se a pontuação de uma métrica do código do aluno for igual ou maior do que o dobro da pontuação máxima aceitável, a métrica do algoritmo do aluno receberá pontuação zero. Desta forma, se a pontuação da métrica do algoritmo do aluno for maior do que a pontuação máxima aceitável e menor do que o dobro da pontuação máxima aceitável, esta métrica receberá uma pontuação equivalente, que será multiplicada pelo peso da métrica. Será apresentado agora um exemplo prático do funcionamento deste esquema de avaliação automática em função das métricas pré-estabelecidas. Um problema armazenado na base de dados do AIIP (Problema X), e uma suposta solução desenvolvida por um aluno para o problema em questão serão utilizados. Os dados referentes ao problema são:
Enunciado: *"O preço de um automóvel é calculado pela soma do preço de fábrica com o preço dos impostos (45% do preço de fábrica) e a percentagem do revendedor (25% do preço de fábrica). Faça um algoritmo que leia o nome do automóvel e o preço de fábrica e imprima o nome do automóvel e o preço final."*

Algoritmo-Resposta:

Algoritmo 5.5 Algoritmo-Resposta do Problema X.

```
Variaveis
    caractere nome
    numerico Precofabrica, PrecoFinal, imposto
Inicio
    Escrever "Digite o Nome do Automóvel:"
    Ler nome
    Escrever "Digite o preco de fábrica do Automóvel:"
    Ler Precofabrica
    imposto := Precofabrica * (0.45 + 0.25)
    PrecoFinal := PrecoFabrica + imposto
    Escrever "Automóvel: ", nome
    Escrever "Preco de Venda: ", PrecoFinal
Fim
```

Analisando o algoritmo-resposta cadastrado para o problema apresentado (Algoritmo 5.5) e considerando os valores contidos na Tabela 5.1, chega-se à pontuação das métricas do Problema X, apresentada na Tabela 5.4. É importante lembrar que os valores de cada métrica são calculados e armazenados para cada problema no momento em que eles são cadastrados no AIIP pelo professor através do Ambiente de Gerenciamento.

Tabela 5.4: Pontuação das métricas do Problema X.

Métrica	**Quantidade**	**Pontuação das Métricas do Algoritmo-Resposta do Problema X**
Quantidade de Linhas de Código	13	13
Quantidade de Variáveis Declaradas	4	8
Quantidade de Instruções	11	22
Quantidade de Estruturas Condicionais	0	0
Quantidade de Estruturas de Repetição	0	0
Quantidade de Funções	0	0

Fonte: Próprio autor.

Analisando o algoritmo desenvolvido por um aluno (Algoritmo 5.6), e com base nos valores contidos na Tabela 5.1, chega-se à pontuação das métricas referentes à solução algorítmica do aluno para o Problema X, apresentadas na Tabela 5.5.

De posse destes valores, o assistente inteligente utiliza as pontuações das métricas relacionadas ao algoritmo-resposta do problema para realizar os cálculos e atribuir uma nota à solução algorítmica desenvolvida pelo do aluno para o Problema X, conforme pode-se observar na Tabela 5.6, onde: A = Pontuação das Métricas do Problema X, B = Pontuação Mínima Aceitável, C = Pontuação Máxima Aceitável, D = Pontuação das Métricas do Algoritmo do Aluno, E = Nota das Métricas do Algoritmo do Aluno, F = Peso na Nota Final e G = Nota Final do Algoritmo do Aluno.

Algoritmo 5.6 Exemplo de Algoritmo desenvolvido por um aluno para o Problema X.

```
Variaveis
    caractere NomeCarro
    numerico PrecoOriginal, PrecoFinal, ImpostoFabrica, PercentagemRevendedor
Inicio
    Escrever "Digite o nome do carro:"
    Ler NomeCarro
    Escrever "Digite o preço original de fábrica do carro:"
    Ler PrecoOriginal
    ImpostoFabrica := PrecoOriginal * (0.45)
    PercentagemRevendedor := PrecoOriginal * (0.25)
    PrecoFinal := PrecoOriginal + ImpostoFabrica + PercentagemRevendedor
    Escrever "O Carro é: ", NomeCarro
    Escrever "O preço de venda do ", NomeCarro, " é: ", PrecoFinal
Fim
```

Tabela 5.5: Pontuação das métricas do algoritmo desenvolvido por um aluno para o Problema X.

Métrica	**Quantidade**	**Pontuação das Métricas do Algoritmo do Aluno para o Problema X**
Quantidade de Linhas de Código	14	14
Quantidade de Variáveis Declaradas	5	10
Quantidade de Instruções	13	26
Quantidade de Estruturas Condicionais	0	0
Quantidade de Estruturas de Repetição	0	0
Quantidade de Funções	0	0

Fonte: Próprio autor.

Tabela 5.6: Formação da nota final do fator algoritmo para o Problema X.

Métrica	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Quantidade de Linhas de Código	13	10,4	15,6	14	10,00	5%	0,50
Quantidade de Variáveis Declaradas	8	5,6	10,4	10	10,00	10%	1,00
Quantidade de Instruções	22	15.4	28.6	26	10,00	10%	1,00
Quantidade de Estruturas Condicionais	0	0	0	0	10,00	30%	3,00
Quantidade de Estruturas de Repetição	0	0	0	0	10,00	20%	2,00
Quantidade de Funções	0	0	0	0	10,00	25%	2,50
Nota Final da Solução Algorítmica do Aluno							**10,00**

Fonte: Próprio autor.

O assistente busca a pontuação de cada métrica do algoritmo-resposta cadastrado para o problema. A seguir, são calculadas as pontuações, mínima e máxima, aceitáveis de acordo com os intervalos de aceitação apresentados na Tabela 5.2. Após isso, o assistente analisa a pontuação de cada métrica do algoritmo do aluno e verifica se estas pontuações encontram-se dentro deste intervalo. Caso estas pontuações estejam dentro do intervalo, atribui-se a nota máxima para a métrica. Como pode-se observar na Tabela 5.6, as pontuações de todas as métricas do algoritmo desenvolvido pelo aluno encontram-se dentro dos intervalos de aceitação, logo, atribuiu-se a nota máxima para todas elas. Ao final, a nota de cada métrica é multiplicada pelo seu peso para se chegar à nota final da solução do aluno.

Neste exemplo é possível observar que o algoritmo desenvolvido pelo aluno praticamente não difere do algoritmo-resposta cadastrado, exceto por uma linha de código a mais, uma variável declarada a mais e duas instruções a mais, sendo justificada a nota recebida pela sua solução apresentada.

É claro que alguns fatores das soluções algorítmicas dos alunos como por exemplo, os nomes utilizados para a identificação de cada variável ou os textos escritos entre aspas não são comparados com o algoritmo-resposta cadastrado, primeiramente por que não interferem significativamente no desempenho e no resultado do algoritmo, e também por que se fossem levados em conta seria praticamente impossível de se atingir uma nota máxima em comparação com o algoritmo-resposta cadastrado.

A seguir observa-se outro exemplo deste esquema de avaliação automática. Os dados referentes ao problema são:

Enunciado: *"Escreva um algoritmo que leia 3 valores inteiros (considere que não serão lidos valores iguais) e escreva-os em ordem crescente."*

Algoritmo-Resposta:

Algoritmo 5.7 Algoritmo-Resposta do Problema Y.

```
Variaveis
    numerico a, b, c
Inicio
    Escrever "Digite um valor qualquer: "
    Ler a
    Escrever "Digite outro valor qualquer: "
    Ler b
    Escrever "Digite mais um valor qualquer: "
    Ler c
    Se (a < b) e (a < c) Entao
        Se (b < c) Entao
            Escrever "Números em ordem crescente: ", a, ", ", b, ", ", c
        Senao
            Escrever "Números em ordem crescente: ", a, ", ", c, ", ", b
        Fim_se
    Fim_se
    Se (b < a) e (b < c) Entao
        Se (a < c) Entao
            Escrever "Números em ordem crescente: ", b, ", ", a, ", ", c
        Senao
            Escrever "Números em ordem crescente: ", b, ", ", c, ", ", a
        Fim_se
    Senão
        Se (b < a) Entao
            Escrever "Números em ordem crescente: ", c, ", ", b, ", ", a
        Senao
            Escrever "Números em ordem crescente: ", c, ", ", a, ", ", b
        Fim_se
    Fim_se
Fim
```

Analisando o algoritmo-resposta cadastrado para o problema apresentado (Algoritmo 5.7), utilizando como base os valores contidos na Tabela 5.1, chega-se à pontuação das métricas do Problema Y, apresentadas na Tabela 5.7.

Tabela 5.7: Pontuação das métricas do Problema Y.

Métrica	**Quantidade**	**Pontuação das Métricas do Algoritmo Ideal do Problema Y**
Quantidade de Linhas de Código	30	30
Quantidade de Variáveis Declaradas	3	6
Quantidade de Instruções	19	38
Quantidade de Estruturas Condicionais	5	15
Quantidade de Estruturas de Repetição	0	0
Quantidade de Funções	0	0

Fonte: Próprio autor.

Algoritmo 5.8 Exemplo de Algoritmo desenvolvido por um aluno para o Problema Y.

```
Variaveis
    numerico a
    numerico b
    numerico c
Inicio
    Escrever "Digite o primeiro número:"
    Ler a
    Escrever "Digite o segundo número:"
    Ler b
    Escrever "Digite o terceiro número:"
    Ler c
    Se (a < b) e (b < c) Entao
        Escrever "Números em ordem crescente: ', a, ', ', b, ', ', c"
    Fim_se
    Se (b < a) e (a < c) Entao
        Escrever ('Números em ordem crescente: ', b, ', ', a, ', ', c"
    Fim_se
    Se (c < a) e (a < b) Entao
        Escrever "Números em ordem crescente: ', c, ', ', a, ', ', b"
    Fim_se
    Se (a < c) e (c < b) Entao
        Escrever "Números em ordem crescente: ', a, ', ', c, ', ', b"
    Fim_se
    Se (b < c) e (c < a) Entao
        Escrever "Números em ordem crescente: ', b, ', ', c, ', ', a"
    Fim_se
    Se (c < b) e (b < a) Entao
        Escrever "Números em ordem crescente: ', c, ', ', b, ', ', a"
    Fim_se
    Se (a < b) e (b > c) e (a < c) Entao
        Escrever "Números em ordem crescente: ', a, ', ', c, ', ', b"
    Fim_se
    Se (a < b) e (b > c) e (a > c) Entao
        Escrever "Números em ordem crescente: ', c, ', ', a, ', ', b"
    Fim_se
    Se (b < a) e (a > c) e (b < c) Entao
        Escrever "Números em ordem crescente: ', b, ', ', c, ', ', a"
    Fim_se
    Se (b < a) e (a > c) e (b > c) Entao
        Escrever "Números em ordem crescente: ', c, ', ', b, ', ', a"
    Fim_se
    Se (a < c) e (b < c) e (a < b) Entao
        Escrever "Números em ordem crescente: ', a, ', ', b, ', ', c"
    Fim_se
    Se (a < c) e (b < c) e (a > b) Entao
        Escrever "Números em ordem crescente: ', b, ', ', a, ', ', c"
    Fim_se
Fim
```

Analisando agora o algoritmo desenvolvido pelo aluno (Algoritmo 5.8), chega-se à pontuação das métricas referentes à solução algorítmica do aluno para o Problema Y, apresentadas na Tabela 5.8.

Tabela 5.8: Pontuação das métricas do algoritmo desenvolvido por um aluno para o Problema Y.

Métrica	**Quantidade**	**Pontuação das Métricas do Algoritmo do Aluno para o Problema Y**
Quantidade de Linhas de Código	48	48
Quantidade de Variáveis Declaradas	3	6
Quantidade de Instruções	48	96
Quantidade de Estruturas Condicionais	12	36
Quantidade de Estruturas de Repetição	0	0
Quantidade de Funções	0	0

Fonte: Próprio autor.

De posse destes valores, o assistente inteligente então utiliza as pontuações das métricas relacionadas ao algoritmo-resposta cadastrado para o problema para realizar os cálculos e atribuir uma nota à solução algorítmica desenvolvida pelo do aluno para o Problema Y, conforme pode-se observar na Tabela 5.9, onde: A = Pontuação das Métricas do Problema Y, B = Pontuação Mínima Aceitável, C = Pontuação Máxima Aceitável, D = Pontuação das Métricas do Algoritmo do Aluno, E = Nota das Métricas do Algoritmo do Aluno, F = Peso na Nota Final e G = Nota Final do Algoritmo do Aluno.

Tabela 5.9: Formação da nota final do fator algoritmo para o Problema Y.

Métrica	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Quantidade de Linhas de Código	30	24	36	48	6,66	5%	0,33
Quantidade de Variáveis Declaradas	6	4,2	7,8	6	10,00	10%	1,00
Quantidade de Instruções	38	26,6	49,4	96	0,56	10%	0,05
Quantidade de Estruturas Condicionais	15	12	18	36	0,00	30%	0,00
Quantidade de Estruturas de Repetição	0	0	0	0	10,00	20%	2,00
Quantidade de Funções	0	0	0	0	10,00	25%	2,50
Nota Final da Solução Algorítmica do Aluno							**5,88**

Fonte: Próprio autor.

Na Tabela 5.9, as pontuações referentes às métricas ***quantidade de linhas de código***, ***quantidade de instruções*** e ***quantidade de estruturas condicionais*** não se encontram dentro dos intervalos de aceitação, logo, foram atribuídas a estas métricas uma nota proporcional de acordo com as Equações 5.1 e 5.2 apresentadas e explicadas anteriormente. Ao final a nota de cada métrica foi multiplicada pelo seu peso para se chegar à nota final da solução algorítmica do aluno.

Neste segundo exemplo é possível observar de uma melhor forma o funcionamento desta metodologia de avaliação, em comparação com a situação anterior apresentada, uma vez que o algoritmo desenvolvido pelo aluno difere bem mais do algoritmo-resposta cadastrado, principalmente com relação às métricas em que suas pontuações não se encontram dentro dos intervalos de aceitação. Desta forma, este algoritmo recebe uma nota proporcional nestas métricas, o que influencia na nota final da solução algorítmica apresentada.

5.4.2 Métricas e Avaliação do Fator Dicas

Para o fator Dicas, as métricas definidas que servirão para a avaliação e atribuição de uma nota para este fator são: nível de conhecimento do aluno, quantidade de dicas da unidade relacionadas com o nível de conhecimento do aluno e a quantidade de dicas apresentadas ao aluno. Quanto maior o nível de conhecimento do aluno, mais penalizado ele é ao solicitar uma dica.

Na Tabela 5.10 são apresentados alguns exemplos de atribuição de nota para o fator Dicas.

Tabela 5.10: Exemplo de atribuição de nota para o fator Dicas.

ID Problema	ID Unidade	Nível de Conhecimento do Aluno	Qnt de Dicas Disponíveis (por Nível)	Qnt de Dicas Apresentadas	Nota
1	1	Insuficiente	30	5	9,17
2	1	Insuficiente	30	10	8,33
3	1	Moderado	30	10	7,33
4	1	Moderado	30	5	8,67
5	2	Insuficiente	30	5	9,17
6	2	Moderado	15	10	4,67

Fonte: Próprio autor.

Como pode-se observar na Tabela 5.10, para os problemas 1 e 2 o nível de conhecimento dos alunos é igual, porém, a nota atribuída no fator Dicas ao aluno do problema 2 foi menor do que para o aluno do problema 1 pelo fato do número de dicas apresentadas ser maior para o aluno do problema 2. Analisando agora os problemas 2 e 3, o número de dicas apresentadas aos aluno foi o mesmo, porém, a nota atribuída no final foi menor para o aluno do problema 3 pelo fato deste possuir um nível de conhecimento maior do que o aluno do problema 2.

Se apenas uma dica for apresentada ao aluno durante o processo em que ele estiver desenvolvendo a solução, independente do seu nível de conhecimento, então ele recebe a nota máxima, numa escala de 0 a 10. Esta condição foi aplicada por acreditar-se que o aluno não pode ter sido beneficiado recebendo apenas uma dica, e consequentemente conseguiu submeter sua solução sem uma ajuda extra mais significativa neste fator.

A partir da apresentação de mais de uma dica ao aluno, então o cálculo para atribuição da nota é o seguinte: divide-se a quantidade de dicas apresentadas ao aluno pela quantidade de

dicas disponíveis, de acordo com o nível de conhecimento do aluno. Após isso, multiplica-se o resultado por um fator de exigência correspondente ao nível de conhecimento do aluno, onde, para os níveis insuficiente, moderado e satisfatório os valores são: 5, 7,5 e 10, respectivamente.

Ao final, subtrai-se de 10 o valor encontrado para então obter-se a nota final para o fator Dicas, conforme mostra a Equação 5.3, onde:

N: nota final do fator Dicas.
Da: quantidade de dicas apresentadas ao aluno.
Dd: quantidade de dicas disponíveis para apresentação.
Fe: fator de exigência correspondente ao nível de conhecimento do aluno, onde para o nível insuficiente ***Fe*** = 5, para o nível moderado, ***Fe*** = 7,5 e para o nível satisfatório, ***Fe*** = 10.

$$N = 10 - \left(\frac{Da}{Dd} Fe\right) \tag{5.3}$$

Ainda assim, a apresentação destas dicas aos alunos deve acontecer de forma organizada, ou seja, não é porque o aluno tem uma dúvida que o mesmo pode solicitar uma dica e o sistema apresentar de uma só vez todas as dicas disponíveis para ele naquele momento. Para isso foi elaborado um esquema para o tempo de apresentação das dicas aos alunos, conforme pode-se observar no exemplo apresentado na Tabela 5.11.

Tabela 5.11: Exemplo do tempo de apresentação das Dicas aos alunos.

ID Problema	ID Unidade	Nível de Conhecimento do Aluno	Qnt de Dicas Disponíveis (por Nível)	Tempo Médio de Resolução do Problema	Intervalo de Apresentação das Dicas
1	1	Insuficiente	30	300 seg	10 seg
2	1	Insuficiente	30	240 seg	8 seg
3	1	Moderado	30	300 seg	17,5 seg
4	1	Moderado	30	240 seg	14 seg
5	2	Insuficiente	30	600 seg	20 seg

Fonte: Próprio autor.

Como pode-se observar, os problemas 1 e 3 pertencem à mesma unidade de conhecimento e possuem o mesmo tempo médio de resolução (300 segundos), mas o intervalo entre a apresentação de uma dica para outra é diferente. No problema 1, somente a cada 10 segundos após a apresentação de uma dica, outra dica já estará disponível para ser apresentada ao aluno, enquanto no problema 3 este intervalo entre a apresentação das dicas é de 17,5 segundos. Essa diferença acontece pelo fato do níveis de conhecimento dos alunos serem diferentes, onde o aluno do problema 3 possui um nível de conhecimento maior do que aluno do problema 1.

Se o nível de conhecimento do aluno for Insuficiente, então o cálculo realizado para obter-se o intervalo de apresentação das dicas ao aluno é realizado da seguinte forma: divide-se o tempo médio de resolução do problema pela quantidade de dicas disponíveis para apresentação, conforme pode-se observar na Equação 5.4, onde:

I: intervalo para apresentação das dicas ao aluno.
T: tempo médio de resolução do problema.
Dd: quantidade de dicas disponíveis para apresentação.

$$I = \left(\frac{T}{Dd} \right) \tag{5.4}$$

Porém, se o nível de conhecimento do aluno for Moderado ou Satisfatório, acrescenta-se à Equação 5.4 a multiplicação do tempo médio de resolução do problema por uma porcentagem deste tempo, onde para o nível de conhecimento Moderado, p = 2,5% e para o nível de conhecimento Satisfatório, p = 5%, conforme pode-se observar na Equação 5.5.

$$I = \left(\frac{T}{Dd} + Tp \right) \tag{5.5}$$

Desta forma, foi criada uma metodologia para avaliar o aluno com relação à quantidade de dicas apresentadas ao mesmo. É preciso deixar claro que não necessariamente uma dica apresentada atenderá todas as expectativas do aluno com relação ao apoio na solução do problema em que ele estiver trabalhando. Como já mencionado anteriormente, a ideia é que o aluno tente solucionar os problemas por conta própria, sem nenhum auxílio, mas como nem sempre isto é possível, estas dicas servem então como mais um apoio pedagógico para o aluno no momento de construção da sua solução para o problema, consequentemente, os alunos são penalizados e avaliados por utilizarem deste auxílio.

5.4.3 Métricas e Avaliação do Fator Refinamentos

Para o fator Refinamentos, as métricas definidas que servirão para a avaliação e atribuição de uma nota para este fator são: nível de conhecimento do aluno, quantidade de refinamentos do problema e a quantidade de refinamentos apresentados ao aluno.

O cálculo para atribuição da nota para o fator Refinamentos segue um esquema parecido da atribuição de nota para o fator Dicas. Quanto maior o nível de conhecimento do aluno, mais penalizado ele é ao solicitar um refinamento.

Na Tabela 5.12 são apresentados alguns exemplos de atribuição de nota para o fator Refinamentos.

Tabela 5.12: Exemplo de atribuição de nota para o fator Refinamentos.

ID Problema	Nível de Conhecimento do Aluno	Qnt de Refinamentos	Qnt de Refinamentos Apresentados	Nota
1	Insuficiente	10	4	8,00
2	Insuficiente	8	7	5,63
3	Moderado	10	4	6,80
4	Moderado	20	13	4,80
5	Moderado	6	4	4,67
6	Satisfatório	10	4	6,00
7	Satisfatório	5	1	10,00

Fonte: Próprio autor.

Como pode-se observar, os problemas 1, 3 e 6 possuem a mesma quantidade de refinamentos associados à eles: 10 (dez), e os alunos ao tentarem resolver estes problemas solicitaram a mesma quantidade de refinamentos para estes três problemas: 4 (quatro). Porém, a nota atribuída no fator Refinamentos para estes 3 problemas foi diferente, sendo a maior nota atribuída para o aluno do problema 1 e a menor para o aluno problema 6. Isto acontece pelo fato do nível de conhecimento do aluno do problema 6 ser o maior e o nível de conhecimento do aluno do problema 1 ser o menor dentre eles.

Assim como na análise do fator Dicas, para o fator Refinamentos se apenas um refinamento for apresentado ao aluno durante o processo em que ele estiver desenvolvendo a solução, independente do seu nível de conhecimento, então ele recebe a nota máxima, numa escala de 0 a 10. A partir da apresentação de mais de um refinamento, o cálculo para atribuição da nota é o mesmo utilizado para a atribuição da nota no fator Dicas, alterando apenas as variáveis: divide-se a quantidade de refinamentos apresentados ao aluno pela quantidade de refinamentos disponíveis para apresentação no problema. Após isso, multiplica-se o resultado por um fator de exigência correspondente ao nível de conhecimento do aluno, onde, para os níveis de conhecimento Insuficiente, Moderado e Satisfatório os valores são: 5, 7,5 e 10, respectivamente.

Ao final, subtrai-se de 10 o valor encontrado para então obter-se a nota final para o fator Refinamentos, conforme mostra a Equação 5.6, onde:

N: nota final do fator Refinamentos.
Ra: quantidade de refinamentos apresentados ao aluno.
Rd: quantidade de refinamentos disponíveis para apresentação.
Fe: fator de exigência correspondente ao nível de conhecimento do aluno, onde para o nível de conhecimento Insuficiente, ***Fe*** = 5, para o nível de conhecimento Moderado, ***Fe*** = 7,5 e para o nível de conhecimento Satisfatório, ***Fe*** = 10.

$$N = 10 - \left(\frac{Ra}{Rd} Fe\right) \tag{5.6}$$

Assim como acontece na apresentação das dicas relacionadas ao fator Dicas, a apresentação dos refinamentos também deve acontecer de forma organizada. Para isso, o mesmo esquema de tempo de apresentação das dicas é seguido para o tempo de apresentação dos refinamentos, conforme pode-se observar no exemplo apresentado na Tabela 5.13.

Tabela 5.13: Exemplo do tempo de apresentação dos Refinamentos aos alunos.

ID Problema	**Nível de Conhecimento do Aluno**	**Qnt de Refinamentos**	**Tempo de Resolução do Problema**	**Intervalo de Apresentação dos Refinamentos**
1	Insuficiente	10	300 seg	30 seg
2	Insuficiente	5	240 seg	48 seg
3	Moderado	10	300 seg	37,5 seg
4	Moderado	5	240 seg	54 seg
5	Moderado	6	600 seg	115 seg
6	Satisfatório	10	300 seg	45 seg
7	Satisfatório	5	240 seg	60 seg

Fonte: Próprio autor.

Como pode-se observar, os problemas 1, 3 e 6 possuem a mesma quantidade de refinamentos associados à eles: 10 (dez), e também possuem o mesmo tempo médio de resolução (300 segundos), porém o intervalo entre a apresentação de um refinamento para outro é diferente. No problema 1, somente a cada 30 segundos após a apresentação de um refinamento, um outro refinamento estará disponível para ser apresentado ao aluno, enquanto que, para o problema 3 este intervalo entre a apresentação dos refinamentos é de 37,5 segundos e para o problema 6 o intervalo é de 45 segundos. Essa diferença acontece pelo fato dos alunos possuírem nível de conhecimento diferentes, possuindo o aluno do problema 6 um nível de conhecimento maior do que os outros dois alunos.

Seguindo o raciocínio utilizado para a apresentação das dicas, se o nível de conhecimento do aluno for Insuficiente, então o cálculo realizado para obter-se o intervalo de apresentação dos refinamentos é realizado da seguinte forma: divide-se o tempo médio de resolução do problema pela quantidade disponível de refinamentos deste problema, conforme pode-se observar na Equação 5.7, onde:

I: intervalo para apresentação dos refinamentos ao aluno.
T: tempo médio de resolução do problema.
Rd: quantidade de refinamentos disponíveis para apresentação.

$$I = \left(\frac{T}{Rd} \right) \tag{5.7}$$

Porém, se o nível de conhecimento do aluno for Moderado ou Satisfatório, acrescenta-se à fórmula anterior a multiplicação do tempo médio de resolução do problema por uma porcentagem deste tempo, onde para o nível de conhecimento Moderado, p = 2,5% e para o nível de conhecimento Satisfatório, p = 5%, conforme pode-se observar na Equação 5.8.

$$I = \left(\frac{T}{Rd} + Tp \right) \tag{5.8}$$

Desta forma, foi criada uma metodologia de avaliação do aluno com relação à quantidade de refinamentos apresentados ao mesmo. Diferentemente das dicas, estes refinamentos provavelmente atenderão de forma mais efetiva as expectativas do aluno com relação ao apoio na construção da solução para a resolução do problema em que ele estiver trabalhando, uma vez que estes refinamentos correspondem na verdade aos passos, etapas ou caminhos que devem ser seguidos para que seja construída uma solução que resolva o problema. Como os alunos, principalmente os iniciantes no domínio da programação, muitas vezes apresentam dúvidas no momento da implementação de algoritmos visando solucionar algum problema, esta alternativa pedagógica, assim como as dicas gerais, também serve como apoio ao aluno.

Porém, como a intenção é auxiliar, e não dar as respostas de graça aos alunos, os mesmos são penalizados e avaliados quanto ao uso destas alternativas. Ou seja, quanto mais dicas e refinamentos forem apresentados, mais penalizados serão estes alunos. Consequentemente, a nota final de sua solução algorítmica, independentemente da solução estar completamente correta, sofrerá um decréscimo nestes fatores.

5.4.4 Métricas e Avaliação do Fator Erros

Uma das maiores dificuldades em facilitar a aprendizagem dos usuários menos experientes através de seus erros cometidos é emitir mensagens que sejam simples de compreender. Exibir mensagens informando apenas que houve um erro é bem mais simples do que exibir mensagens informando e sugerindo ao usuário o que fazer naquele caso específico, uma vez que o mesmo estado interno de erro do compilador pode ocorrer para diversos erros diferentes.

Durante o período em que o aluno estiver construindo sua solução visando à resolução de algum problema, o assistente inteligente trabalha em segundo plano com o objetivo de identificar os erros cometidos pelo aluno e, se necessário, fornecer *feedbacks* e listar as

possíveis causas e soluções para os erros encontrados, desta forma, atribuindo uma nota para o último dos fatores a serem analisados para a construção da nota final para a solução algorítmica do aluno, o fator Erros.

Seguindo a metodologia abordada nos fatores Dicas e Refinamentos, quanto maior o nível de conhecimento do aluno, mais penalizado ele é ao cometer erros. Subtende-se desta forma que, na medida em que o aluno vá resolvendo os problemas armazenados, aumentando sua capacidade de raciocínio e, teoricamente, conhecendo e evitando cometer certos tipos de erros mais comuns, ele dever ser mais penalizado do que um aluno que ainda não possui um nível de conhecimento satisfatório.

Na Tabela 5.14 são apresentados alguns exemplos de atribuição de nota para o fator Erros.

Tabela 5.14: Exemplo de atribuição de nota para o fator Erros.

Nível de Conhecimento do Aluno	**Qnt de Erros Cometidos**	**Nota**
Insuficiente	1	10
Insuficiente	10	9
Insuficiente	20	8
Insuficiente	30	7
Insuficiente	40	6
Insuficiente	50 +	5
Moderado	1	10
Moderado	7,5	9
Moderado	15	8
Moderado	22,5	7
Moderado	30	6
Moderado	37,5 +	5
Satisfatório	1	10
Satisfatório	5	9
Satisfatório	10	8
Satisfatório	15	7
Satisfatório	20	6
Satisfatório	25 +	5

Fonte: Próprio autor.

Como pode-se observar na Tabela 5.14, se o aluno comete apenas 1 erro durante o processo em que ele estiver desenvolvendo a solução para o problema, independente do seu nível de conhecimento, então ele recebe a nota máxima, numa escala de 5 a 10.

A partir de mais de um erro cometido pelo aluno, identificado pelo assistente inteligente, o cálculo para atribuição da nota para este fator é o seguinte: utiliza-se um fator de exigência Fe correspondente ao nível de conhecimento do aluno, onde para o nível de conhecimento Insuficiente, Fe = 10, para o nível de conhecimento Moderado, Fe = 7,5 e para o nível de

conhecimento Satisfatório, Fe = 5. A partir disso, verifica-se a quantidade de erros cometidos pelo aluno (E) e divide-se esta quantidade pelo Fe correspondente ao nível de conhecimento do aluno. Após isso, subtrai-se de 10 o valor encontrado e finalmente chega-se à nota final (N) referente ao fator Erros, conforme pode-se observar na Equação 5.9.

$$N = 10 - \left(\frac{E}{Fe}\right) \tag{5.9}$$

Por exemplo, ao cometer 15 erros (E = 15) durante a construção de um problema, um aluno com nível de conhecimento Moderado, ou seja, Fe = 7,5, receberá a nota 8 para este fator. Já para um aluno com nível de conhecimento Satisfatório, ao cometer esta mesma quantidade de erros (E = 15), com Fe = 5, receberá a nota 7 para este fator.

Desta forma, foi criada a metodologia de avaliação do último fator (Erros), levado em consideração para a avaliação e atribuição de uma nota para a solução algorítmica do aluno objetivando a resolução dos problemas propostos. Assim como os outros 3 fatores analisados: **Algoritmo**, **Dicas** e **Refinamentos**, o fator **Erros** tem sua importância para avaliação das soluções algorítmicas do aluno não apenas por avaliar a quantidade de erros cometidos, mas também por tentar mostrar aos alunos as possíveis causas destes erros através de *feedbacks*, bem como, apresentar possíveis soluções para os erros encontrados.

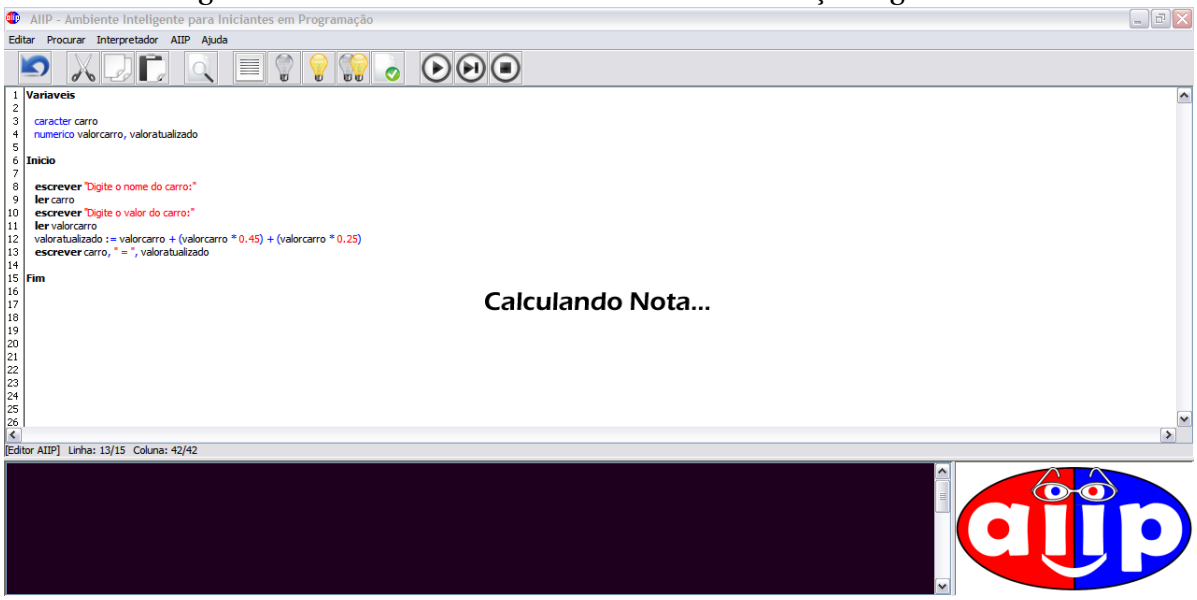
Entende-se que os erros cometidos pelos alunos devem ser passíveis de penalização, independentemente do nível de tolerância aplicável para tal. Por esse motivo o fator Erros foi incluído como um dos itens analisados para se atribuir a nota final para a avaliação da solução algorítmica do aluno.

5.5 Calculando a Nota Final da Solução do Aluno para um Problema

Após atribuir uma nota para cada um dos fatores analisados: o Algoritmo do aluno, os Erros cometidos e as Dicas e Refinamentos solicitados, o assistente é ativado para calcular uma nota final para a solução do aluno (Figuras 5.14 e 5.15), englobando todos estes fatores.

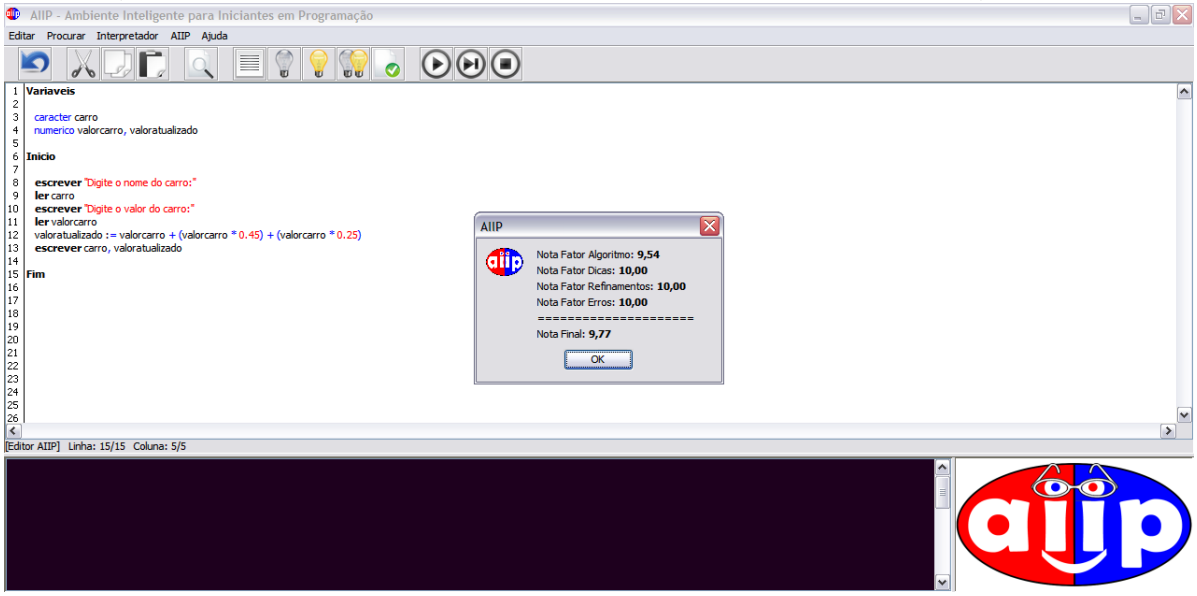
Para cada problema armazenado na base de dados o professor deve atribuir um peso aos fatores analisados para a formação da nota final da solução do aluno. O professor pode levar em consideração o nível de dificuldade do problema, a classe do problema, a probabilidade da ocorrência de erros ou outras características para atribuir um peso para cada fator. No entanto, os pesos de cada fator para um problema apresentados na Tabela 5.15 serão utilizados como exemplo.

Figura 5.14: Assistente calculando a nota da solução algorítmica.



Fonte: Próprio autor.

Figura 5.15: Assistente atribuindo a nota dos fatores e da solução algorítmica.



Fonte: Próprio autor.

Tabela 5.15: Peso dos fatores na nota final de um problema.

Fator	**Peso**
Algoritmo	50%
Refinamentos	25%
Dicas	15%
Erros	10%
Total	**100%**

Fonte: Próprio autor.

É possível observar que o maior peso é atribuído para o fator Algoritmo, onde 50% da nota final da solução do aluno diz respeito a este fator e suas métricas analisadas. O segundo maior peso atribuído é o do fator Refinamentos, onde 25% da nota final para a solução do aluno é calculada de acordo com a nota deste fator. O fator Dicas representa 15% da nota final para a solução do aluno. Por fim, o fator Erros representa os 10% restantes para compor a nota final da solução do aluno.

A nota final para a solução do aluno é obtida através da Equação 5.10, onde:

N: nota final da solução do aluno.
Na: nota do fator Algoritmo.
Nr: nota do fator Refinamentos.
Nd: nota do fator Dicas.
Ne: nota do fator Erros.
Pa: peso do fator Algoritmo.
Pr: peso do fator Refinamentos.
Pd: peso do fator Dicas.
Pe: peso do fator Erros.

$$N = (Na \times Pa) + (Nr \times Pr) + (Nd \times Pd) + (Ne \times Pe) \tag{5.10}$$

Para exemplificar a atribuição destes pesos e como a nota de cada fator pode influenciar na nota final de uma solução será simulada uma situação onde um aluno submeteu sua solução e obteve a seguinte nota em cada uma dos fatores a seguir: **Algoritmo** = 6,0, **Refinamentos** = 8,0, **Dicas** = 10,0 e **Erros** = 9,0. Utilizando os pesos apresentados Tabela 5.15, calcula-se então que a nota final da solução do aluno para esta simulação seria igual a 7,40, conforme pode-se observar Tabela 5.16.

Tabela 5.16: Exemplo de nota atribuída para um problema.

Fator	**Nota do Fator**	**Peso do Fator**	**Nota Final**
Algoritmo	6,0	50%	3,0
Refinamentos	8,0	25%	2,0
Dicas	10,0	15%	1,5
Erros	9,0	10%	0,9
Nota Final			**7,4**

Fonte: Próprio autor.

No exemplo apresentado, mesmo considerando como insatisfatória a nota do fator Algoritmo (6,0), o aluno recebeu uma nota final para sua solução que pode ser considerada como satisfatória (acima de 7), compensado pelas notas dos outros fatores.

5.6 Avançando de Unidade e Resolvendo Novos Problemas

Uma vez calculada a nota final para a solução do aluno em resposta a um problema, o aluno pode então tentar solucionar outros problemas associados à unidade atual que ele está estudando, visando atingir a pontuação mínima necessária para avançar de unidade, estudar conceitos mais avançados, e, consequentemente, resolver os problemas associados a estas unidades subsequentes.

O requisito para que o aluno consiga avançar de unidade é simples: ele deve atingir, no mínimo, 70% da pontuação máxima esperada pela resolução de todos os problemas associados à unidade. Como cada unidade de conhecimento possui associada à ela diversos problemas, e estes problemas possuem um nível de dificuldade (1, 2 ou 3), a pontuação máxima esperada pela resolução de todos os problemas de cada unidade é calculada multiplicando-se o valor 10, correspondente à nota máxima possível para todos os problemas, pela quantidade de problemas de cada nível (1, 2 e 3) e pelo peso correspondente ao nível dos problemas, onde para os problemas com nível de dificuldade 1 e 2 o peso associado é $p = 3$ e para os problemas com nível de dificuldade 3 o peso associado é $p = 4$. Após isso, somam-se os valores encontrados para se chegar à pontuação máxima da unidade.

Observa-se na Tabela 5.17 uma simulação para identificar a pontuação mínima necessária para que o aluno possa avançar de unidade e habilitar-se a resolver os problemas associados a uma unidade mais avançada, onde: A = Quantidade de Problemas, B = Quantidade de Problemas Nível 1, C = Peso dos Problemas Nível 1, D = Quantidade de Problemas Nível 2, E = Peso dos Problemas Nível 2, F = Quantidade de Problemas Nível 3, G = Peso dos Problemas Nível 3, H = Pontuação Máxima na Unidade e I = Pontuação Mínima na Unidade.

Tabela 5.17: Exemplo do cálculo da pontuação máxima de cada unidade.

Unidade	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
1	20	10	3	5	3	5	4	650	455
2	12	5	3	4	3	3	4	390	273
3	15	6	3	5	3	4	4	490	343

Fonte: Próprio autor.

Na Tabela 5.17 observa-se a simulação de 3 unidades de conhecimento e a quantidade de problemas associados à elas, onde para a unidade 1, a quantidade de problemas associados é 20, para a unidade 2 a quantidade de problemas associados é 12 e para a unidade 3 a quantidade de problemas associados é 15.

Tomando como exemplo a unidade 1, observa-se que dos 20 problemas associados à ela, 10 possuem nível de dificuldade 1, 5 possuem nível de dificuldade 2 e os outros 5 nível de dificuldade 3. Multiplicando a quantidade de problemas pelos seus pesos associados, de acordo com seu nível, e, ao final, somando-se os valores encontrados, obtêm-se 650 como sendo a pontuação máxima que o aluno pode alcançar na unidade 1. Como o requisito para

que o aluno consiga avançar de unidade é que ele deve atingir, no mínimo, 70% da pontuação máxima esperada pela resolução de todos os problemas associados à unidade, então, ao alcançar 455 de pontuação, o aluno está liberado para avançar à unidade 2 e resolver os problemas associados à essa unidade.

É importante lembrar que a nota obtida pelo aluno em cada problema é armazenada na base de dados, onde o assistente inteligente verifica sempre, ao final de cada resolução de problema, se o aluno já tem condições de avançar de unidade e de resolver os problemas das unidades subsequentes.

6 AVALIAÇÃO DO AIIP

Através da realização de uma avaliação formal é possível adotar providências para aperfeiçoar ou contornar problemas relacionados ao desenvolvimento de um sistema. Como nem sempre os atributos de um sistema podem ser avaliados de foma direta, a Engenharia de *Software* adota métricas com o objetivo de atribuir valores e avaliar, ainda que de forma subjetiva, se estes atributos estão correspondendo aos seus objetivos.

Neste sentido, com o objetivo de avaliar o ambiente proposto neste trabalho, realizou-se um experimento com 21 alunos que já cursaram a disciplina de Algoritmos e Estrutura de Dados I do curso de Sistemas de Informação à Distância da Universidade Federal de Alagoas. A princípio, esta avaliação seria realizada com alunos do curso de Ciência da Computação da mesma instituição, mas devido a uma greve de quase 4 meses e da aproximação da data de apresentação deste trabalho, optou-se pela avaliação com os alunos do curso de Sistemas de Informação.

O experimento foi dividido em três etapas (**Planejamento, Teste Prático e Aplicação do Questionário e Análise dos Dados**). As seções seguintes apresentam o processo de desenvolvimento do experimento, bem como os resultados obtidos com o mesmo.

6.1 Etapa 1: Planejamento

Na etapa do planejamento foram identificados os objetivos almejados pelo ambiente proposto, determinadas as questões para verificar o alcance destes objetivos e definidas as métricas que serviriam de base para responder cada questão, seguindo a metodologia GQM Basili et al. (1994), que consequentemente guiou a elaboração do questionário (Apêndice B) aplicado aos alunos.

Os objetivos identificados, a serem alcançados com a implementação do ambiente proposto, foram os seguintes:

- **Objetivo 1 (Geral)**: Implementar um ambiente para auxiliar alunos e professores no processo de ensino/aprendizado de programação.
- **Objetivo 2 (Específico)**: Prover uma boa usabilidade para o ambiente.
- **Objetivo 3 (Específico)**: Prover flexibilidade no gerenciamento dos recursos do ambiente.

Após a identificação destes objetivos foi possível determinar as questões que poderiam responder, de forma quantitativa, se estes objetivos foram alcançados com o desenvolvimento do ambiente. A seguir, cada objetivo é explorado individualmente com relação às suas questões e métricas.

Objetivo 1: Implementar um ambiente para auxiliar alunos e professores no processo de ensino/aprendizado de programação.

Questão 1: Qual o índice de aprovação dos recursos pedagógicos oferecidos pelo ambiente?

Métricas:

- Suficiência das funcionalidades e recursos pedagógicos oferecidos.
- Limitações das funcionalidades e recursos pedagógicos oferecidos.
- Concordância com a utilização da resolução de problemas como estratégia pedagógica.
- Satisfação com a metodologia de avaliação automática.
- Importância do Assistente Inteligente.
- Importância do Ambiente Teórico de ensino.
- Relevância do simulador de execução passo-a-passo do código.

Questão 2: Qual o índice de aprovação dos recursos de ajuda e *feedbacks* oferecidos pelo ambiente?

Métricas:

- Suficiência dos recursos de ajuda e *feedbacks* oferecidos.
- Limitações dos recursos de ajuda e *feedbacks* oferecidos.
- Satisfação com os recursos pedagógicos de ajuda (Dicas e Refinamentos) recebidos durante a resolução dos problemas.
- Satisfação com os *feedbacks* recebidos sobre os erros cometidos.
- Relevância do Ambiente de Estatísticas sobre desempenho.

Questão 3: O ambiente possui capacidade para ser aproveitado no processo de ensino/aprendizado de programação?

Métricas:

- Concordância com a utilização do ambiente para auxílio do aprendizado de alunos iniciantes em programação.
- Concordância com a utilização do ambiente para auxílio nas tarefas de ensino do professor.

Objetivo 2: Prover uma boa usabilidade para o ambiente.

Questão 1: Qual o nível de satisfação com a usabilidade do ambiente e de seus recursos?

Métricas:

- Dificuldade de utilização dos recursos de edição, compilação e execução (normal e passo-a-passo) dos algoritmos.
- Organização dos recursos e funcionalidades do ambiente.
- Limitação dos recursos e funcionalidades do ambiente.

Questão 2: Qual o nível de compreensão da linguagem utilizada pelo ambiente?

Métricas:

- Dificuldade de entendimento da sintaxe da linguagem.
- Simplicidade da linguagem.
- Limitação da linguagem.

Objetivo 3: Prover flexibilidade no gerenciamento dos recursos do ambiente.

Questão 1: Qual o índice de aprovação do gerenciamento dos recursos do ambiente?

Métricas:

- Importância do Ambiente de Gerenciamento.
- Satisfação com a flexibilidade de gerenciamento dos recursos do ambiente.
- Limitação do gerenciamento dos pesos e pontuações das métricas para avaliação automática dos algoritmos.
- Limitação do gerenciamento do material instrucional do ambiente.

6.2 Etapa 2: Teste Prático

Após a definição dos objetivos, questões e métricas, deu-se início à próxima etapa do experimento para avaliação do ambiente: o teste prático.

Nesta etapa os alunos utilizaram o ambiente com o intuito de testar e verificar possíveis falhas no funcionamento do mesmo. Antes, no entanto, os mesmos receberam um tutorial

com informações sobre como proceder para que o ambiente pudesse funcionar de forma satisfatória. Após o entendimento das funcionalidades do ambiente, os alunos utilizaram o mesmo durante um período de 4 horas para praticar e testar, onde neste período as eventuais dúvidas eram esclarecidas em tempo real.

6.3 Etapa 3: Aplicação do Questionário e Análise dos Dados

Ao final da etapa anterior os alunos foram submetidos a um questionário (Apêndice B), elaborado com base nas métricas definidas na etapa do planejamento.

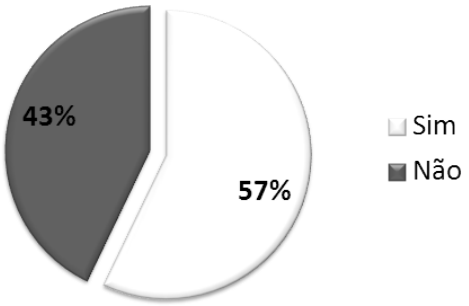
Após a aplicação dos questionários os dados foram tabulados e analisados utilizando-se o *software* estatístico SPSS 20.0, com o propósito de verificar o nível de alcance dos objetivos identificados na etapa do planejamento. A análise possuiu caráter univariado, apesar de terem sidos realizadas algumas análises bivariadas a fim de verificar possíveis correlações entre as variáveis. O resultado da tabulação e análise dos dados, juntamente com os comentários e conclusões, podem ser observados na seção a seguir.

6.4 Resultado da Análise dos Dados

Nesta seção o resultado da tabulação e análise dos dados é apresentado, juntamente com alguns comentários e conclusões obtidas.

Na Figura 6.1 é possível observar o resultado da seguinte questão apresentada aos alunos: "Você costuma utilizar ou já utilizou alguma IDE comercial e/ou profissional para desenvolver seus algoritmos?".

Figura 6.1: Percentual de alunos que costuma utilizar ou já utilizou alguma IDE comercial e/ou profissional para desenvolver seus algoritmos.



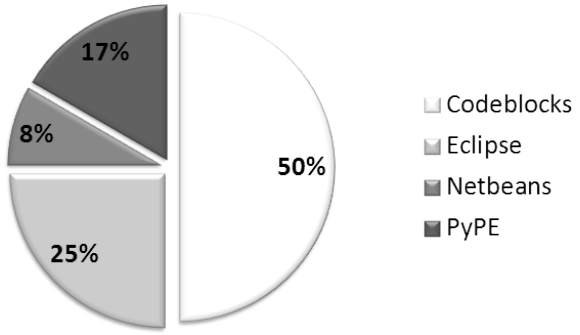
Fonte: Próprio autor.

O propósito desta questão foi identificar o percentual de alunos que possui, no mínimo, familiaridade com alguma IDE comercial e/ou profissional para o desenvolvimento de seus algoritmos, seja com propósito profissional ou educacional, além de servir como base para a

análise de outras questões adiante. Como é possível observar, 57% dos alunos afirmaram utilizar, ou já ter utilizado algum IDE, enquanto 43% afirmaram não utilizar destas ferramentas para o desenvolvimento de seus algoritmos.

Na Figura 6.2 estão representadas as IDE comerciais e/ou profissionais utilizadas para o desenvolvimento dos algoritmos pelos alunos que responderam "Sim" à questão anterior. Destes, 50% utilizam o Codeblocks, 25% utilizam o Eclipse, 17% utilizam o Pype e 8% utilizam o Netbeans. O resultado desta questão é importante no sentido de comparar características do AIIP com algumas características destas IDE sob a perspectiva dos alunos, já que é possível admitir que os mesmos possuem familiaridade com a ferramenta indicada, sendo justificável suas respostas em algumas questões adiantes.

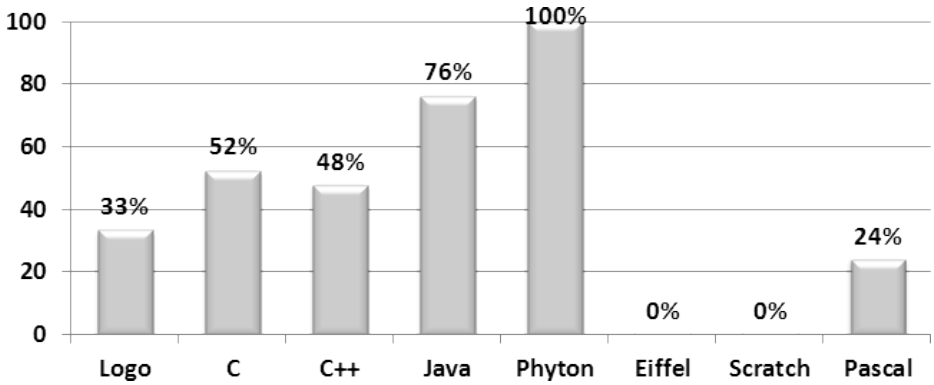
Figura 6.2: IDE utilizadas pelos alunos para o desenvolvimento de seus algoritmos.



Fonte: Próprio autor.

É possível observar na Figura 6.3 o resultado da seguinte questão: "Das linguagens listadas abaixo, assinale as que você, pelo menos, já ouviu falar".

Figura 6.3: Linguagens de familiaridade ou conhecidas pelos alunos.



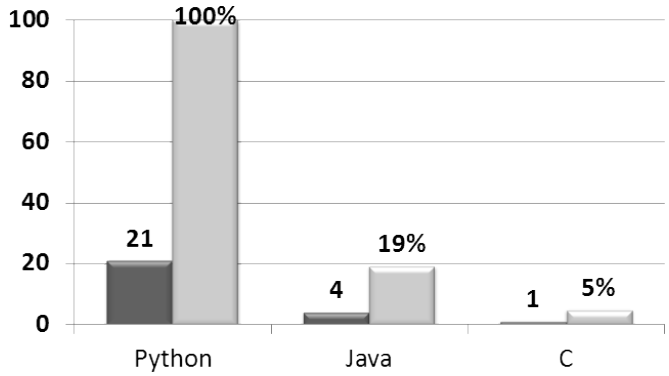
Fonte: Próprio autor.

O propósito desta questão foi identificar as linguagens as quais os alunos possuem mais familiaridade, ou, pelo menos, conhecimento da existência. É possível observar que as linguagens de propósito educacional como por exemplo: Logo, Eiffel, Scratch e Pascal não

são linguagens familiares aos alunos pesquisados, enquanto outras linguagens com propósito mais comercial, como Phyton e Java, por exemplo, são de maior conhecimento por parte destes alunos.

Na Figura 6.4 é possível observar realmente que os alunos participantes do experimento possuem mais familiaridade com as linguagens de propósito comercial, uma vez que 100% destes afirmaram desenvolver ou já ter desenvolvido algoritmos em Phyton, 19% afirmaram desenvolver ou já ter desenvolvido algoritmos em Java e 5% afirmaram desenvolver ou já ter desenvolvido algoritmos em C.

Figura 6.4: Linguagens utilizadas pelos alunos para o desenvolvimento de seus algoritmos.

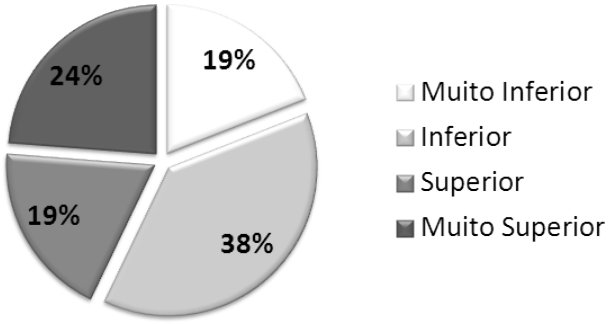


Fonte: Próprio autor.

Percebe-se então que as linguagens de propósito educacional não fazem parte da rotina acadêmica destes alunos, e, apesar do tamanho da amostra ser pequeno, comprova-se o que há algum tempo tem se percebido na área: as linguagens de propósito educacional vem perdendo espaço para outras linguagens no meio acadêmico.

Partindo para a análise das questões que envolvem o AIIP diretamente, observa-se na Figura 6.5 o resultado da seguinte questão: “Em comparação com alguma IDE já utilizada por você, com relação aos seus recursos e funcionalidades de forma geral, o AIIP pode ser considerado”.

Figura 6.5: Comparação do AIIP com as IDE utilizadas pelos alunos.



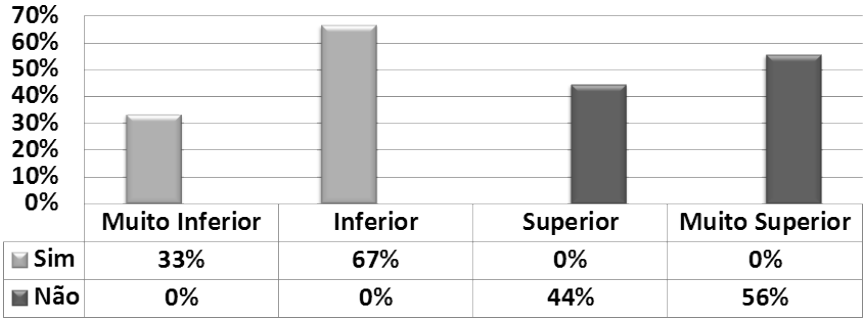
Fonte: Próprio autor.

Conforme está representado, 38% afirmaram que o AIIP pode ser considerado inferior, 24% afirmaram que o AIIP pode ser considerado muito superior, 19% afirmaram que o AIIP pode ser considerado muito inferior e outros 19% afirmaram que o AIIP pode ser considerado superior.

Aparentemente o resultado para esta questão foi dividido. No entanto, é possível entender melhor a resposta para esta questão dividindo os alunos entre aqueles que utilizam, ou já utilizaram alguma IDE para desenvolver seus algoritmos, e aqueles que não utilizam ou nunca utilizaram destas ferramentas, considerando as IDE como editores de código simples para este segundo grupo de alunos.

Desta forma, é possível observar que aqueles que costumam utilizar alguma IDE consideram o AIIP inferior ou muito inferior às ferramentas que eles costumam utilizar, enquanto o restante dos alunos consideram o AIIP superior ou muito superior aos seus editores de código, conforme pode-se observar na Figura 6.6.

Figura 6.6: Comparação do AIIP com as IDE utilizadas pelos alunos, dividida entre os que utilizam ou já utilizaram, e os que não utilizam ou nunca utilizaram destas ferramentas.



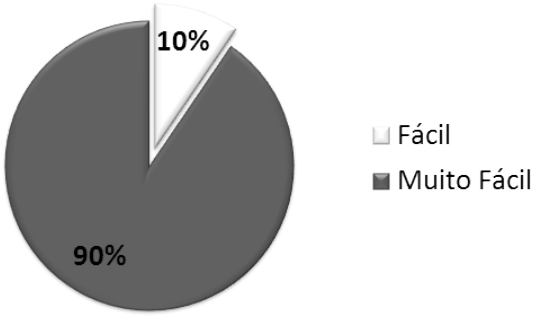
	Muito Inferior	Inferior	Superior	Muito Superior
Sim	33%	67%	0%	0%
Não	0%	0%	44%	56%

Fonte: Próprio autor.

Pode-se admitir que o resultado referente ao primeiro grupo de alunos já era esperado, tratando-se o AIIP de um protótipo inicial, a sua equivalência com ferramentas de desenvolvimento comerciais e/ou profissionais não era esperada, no entanto, a aprovação por parte do segundo grupo de alunos é algo a ser comemorado, pois representa que os recursos e funcionalidades desenvolvidos atendem as necessidades e expectativas daqueles alunos que não se utilizam de ferramentas mais sofisticadas voltadas para o aprendizado.

Com relação aos recursos básicos (edição, compilação e execução), é possível perceber na Figura 6.7 a facilidade de utilização dos mesmos através da aprovação dos alunos, onde 90% afirmaram que a utilização destes recursos no AIIP pode ser considerada muito fácil e 10% afirmaram que a utilização destes recursos pode ser considerada fácil.

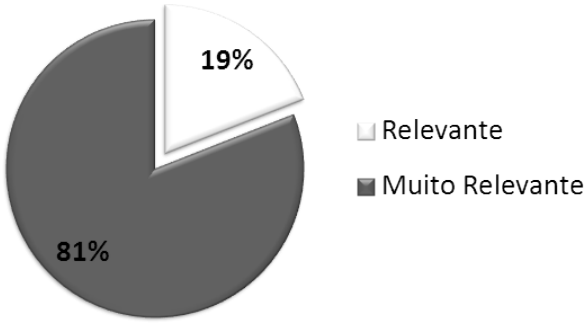
Figura 6.7: Nível de dificuldade de utilização dos recursos básicos de edição, compilação e execução dos algoritmos do AIIP.



Fonte: Próprio autor.

Com relação à relevância do simulador de execução passo-a-passo de códigos do AIIP, também é possível observar na Figura 6.8 uma aprovação por parte dos alunos, uma vez que 81% afirmaram que o mesmo é muito relevante e 19% afirmaram que o mesmo é relevante dentro do AIIP.

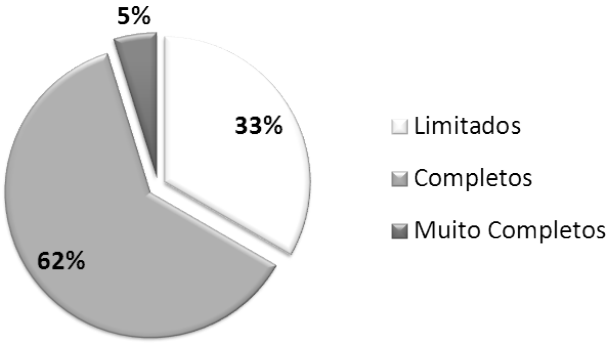
Figura 6.8: Nível de relevância do simulador de execução passo-a-passo de códigos do AIIP.



Fonte: Próprio autor.

A Figura 6.9 representa a opinião dos alunos, de forma geral, em relação a limitação dos recursos e funcionalidades do AIIP.

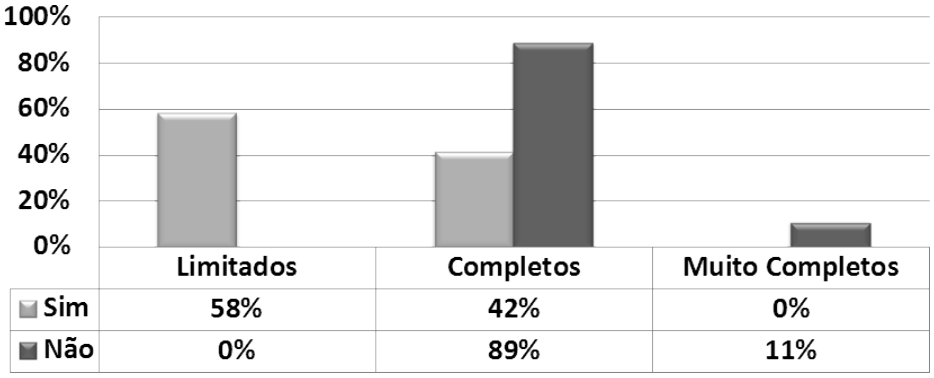
Figura 6.9: Limitação dos recursos e funcionalidades do AIIP.



Fonte: Próprio autor.

Observa-se que 62% afirmaram que estes recursos são muito completos, 33% afirmaram que estes recursos são limitados e 5% afirmaram que estes recursos são muito completos. Esta divisão de opiniões também pode ser justificada pela divisão dos alunos entre aqueles que utilizam, ou já utilizaram alguma IDE para desenvolver seus algoritmos, e aqueles que não utilizam ou nunca utilizaram destas ferramentas, conforme observa-se na Figura 6.10. Para o primeiro grupo, os recursos do AIIP são em sua maioria limitados, enquanto para o segundo grupo os recursos apresentados pelo AIIP podem ser considerados completos.

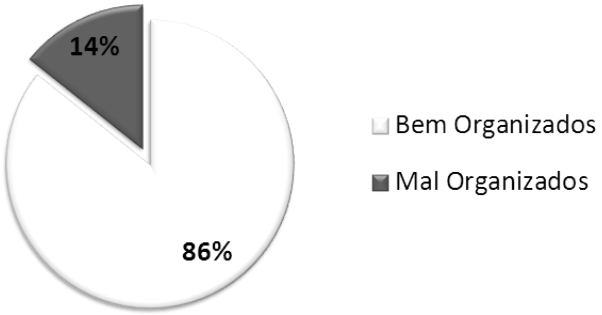
Figura 6.10: Limitação dos recursos e funcionalidades do AIIP, dividida entre os alunos que utilizam ou já utilizaram alguma IDE para o desenvolvimento de seus algoritmos, e os que não utilizam ou nunca utilizaram destas ferramentas.



Fonte: Próprio autor.

Levando em consideração a organização (layout), em um âmbito geral, dos recursos e funcionalidades do AIIP, observa-se na Figura 6.11 que 86% dos alunos afirmaram que estes recursos e funcionalidades são bem organizados, enquanto 14% afirmaram que são mal organizados. Mais uma vez, vale ressaltar que tratando-se o AIIP de um protótipo inicial, este resultado deve ser considerado importante, na medida em que reflete a aprovação da distribuição organizacional destes recursos e funcionalidades sob a perspectiva dos alunos.

Figura 6.11: Nível de organização dos recursos e funcionalidades do AIIP.



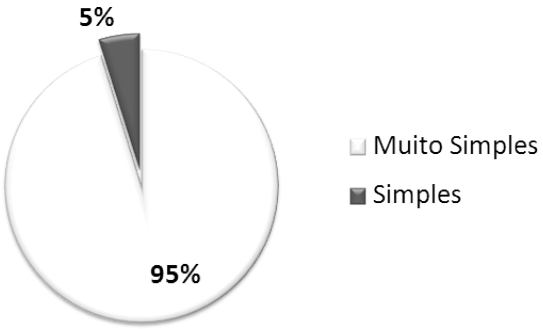
Fonte: Próprio autor.

Analisando alguns aspectos relacionados à linguagem utilizada pelo AIIP (baseada no

ILA), observou-se que para 100% dos alunos, o nível de dificuldade de entendimento da sintaxe da linguagem foi considerado muito fácil, o que alicerça a ideia de utilizar o AIIP em turmas iniciantes de programação, onde através da utilização de uma linguagem de fácil compreensão é possível diminuir a atenção dos alunos com aspectos da sintaxe da linguagem, aumentando a preocupação dos mesmos com a lógica de programação em si.

Já em relação à simplicidade da linguagem, 95% dos alunos afirmaram que a mesma é muito simples, enquanto 5% afirmaram que a mesma é simples, conforme observa-se na Figura 6.12.

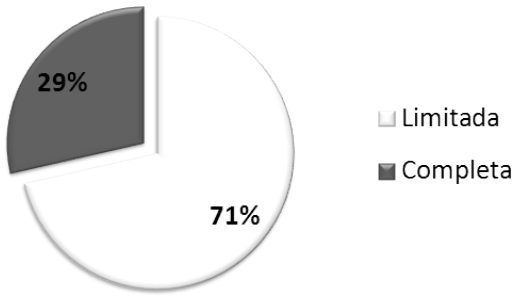
Figura 6.12: Simplicidade da linguagem utilizada no AIIP.



Fonte: Próprio autor.

Em contrapartida, em relação a limitação da linguagem, 71% dos alunos afirmaram que a mesma é limitada, enquanto 29% afirmaram que a mesma é completa, conforme é possível observar na Figura 6.13.

Figura 6.13: Limitação da linguagem utilizada no AIIP.

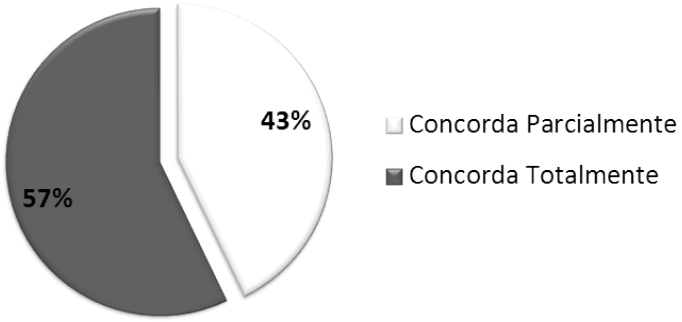


Fonte: Próprio autor.

É importante deixar claro que as questões de simplicidade e limitação da linguagem não estão relacionadas com a sua capacidade computacional, e sim com sua simplicidade de utilização, aprendizado, entendimento, dentre outros aspectos. Além disso, vale destacar que o ILA foi desenvolvido com propósito educacional, baseado no paradigma estrutural, ou seja, alguns recursos mais avançados utilizados por outras linguagens, principalmente linguagens orientadas a objetos, não estão presentes no ILA, e consequentemente no AIIP.

Verificando algumas questões acerca dos aspectos pedagógicos utilizados pelo AIIP, podemos observar na Figura 6.14 o resultado da concordância dos alunos em relação a utilização da resolução de problemas no AIIP como estratégia pedagógica, onde 57% dos alunos afirmaram concordar totalmente, enquanto 43% afirmaram concordar parcialmente. Assim, é possível concluir que o incentivo a resolução de problemas no AIIP obteve uma boa aceitação, apresentando-se como uma boa estratégia no incentivo ao aprendizado de programação.

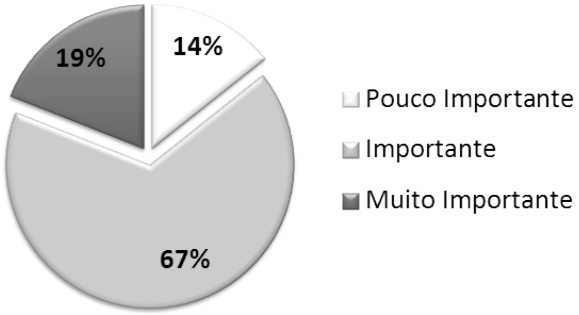
Figura 6.14: Concordância em relação a utilização da resolução de problemas no AIIP como estratégia pedagógica.



Fonte: Próprio autor.

É possível comprovar também a aprovação dos alunos em relação ao assistente inteligente do AIIP, uma vez que 67% consideraram o mesmo importante, 19% consideraram muito importante e apenas 14% consideraram pouco importante para o auxilio no ambiente, conforme pode-se observar na Figura 6.15.

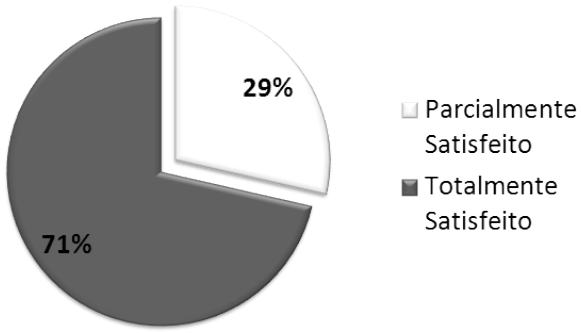
Figura 6.15: Importância do Assistente Inteligente no AIIP.



Fonte: Próprio autor.

Na Figura 6.16 é possível observar o grau de satisfação dos alunos com a metodologia de avaliação automática utilizada no AIIP. Percebe-se que a metodologia de avaliação utilizada também obteve boa aprovação, onde 71% dos alunos afirmaram ter ficado totalmente satisfeitos, enquanto 29% afirmaram ter ficado parcialmente satisfeitos.

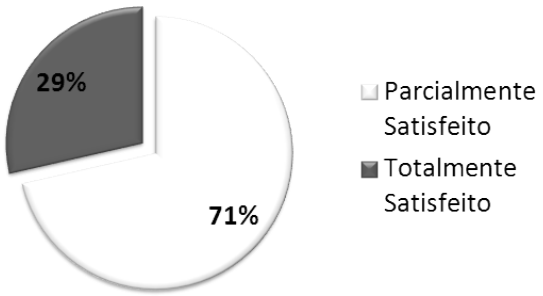
Figura 6.16: Grau de satisfação dos alunos com a metodologia de avaliação automática utilizada no AIIP.



Fonte: Próprio autor.

Na Figura 6.17 está representado o nível de satisfação dos alunos com os *feedbacks* recebidos sobre os erros cometidos durante a interação e resolução dos problemas no AIIP. Mais uma vez observa-se a aprovação dos alunos com relação a este recurso, onde 71% afirmaram estar parcialmente satisfeitos e 29% afirmaram estar totalmente satisfeitos.

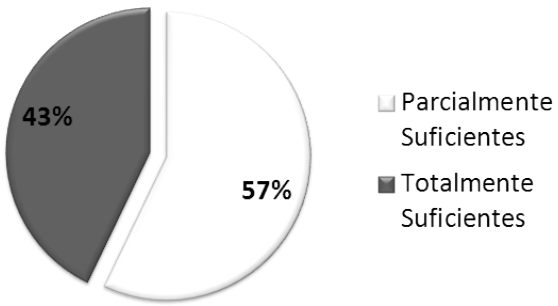
Figura 6.17: Nível de satisfação com os *feedbacks* recebidos sobre os erros cometidos.



Fonte: Próprio autor.

Já em relação à suficiência dos recursos de ajuda (Dicas e Refinamentos) do AIIP, pode-se afirmar que estes recursos também foram bem aceitos pelos alunos.

Figura 6.18: Suficiência dos recursos de ajuda e *feedbacks* do AIIP.



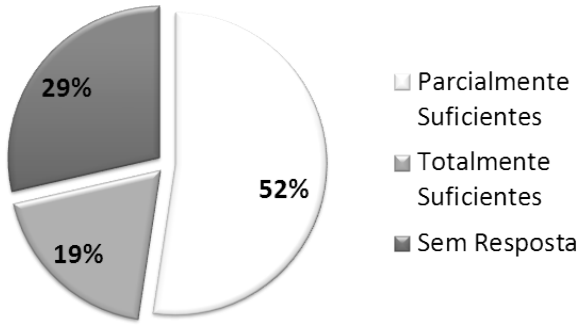
Fonte: Próprio autor.

Conforme é possível observar na Figura 6.18, 57% afirmaram que estes recursos são parcialmente suficientes, enquanto 43% afirmaram que estes recursos são totalmente suficientes.

As Figuras 6.19 e 6.20 representam as únicas 2 questões onde alguns dos alunos não emitiram suas opiniões. Estas questões abordaram sobre os recursos pedagógicos do AIIP em um âmbito geral. Os motivos que levaram alguns alunos a não responderem estas questões podem ser vários: esquecimento, desatenção, não entendimento da questão, etc. No entanto, as respostas válidas representaram a aprovação destes recursos pelos alunos.

Na Figura 6.19 está representado o resultado da seguinte questão: "Para você, as funcionalidades e recursos pedagógicos oferecidos são". Observa-se que 52% dos alunos afirmaram que estes recursos são parcialmente suficientes, enquanto 19% afirmaram que estes recursos são totalmente suficientes, tendo 29% dos alunos não respondido esta questão.

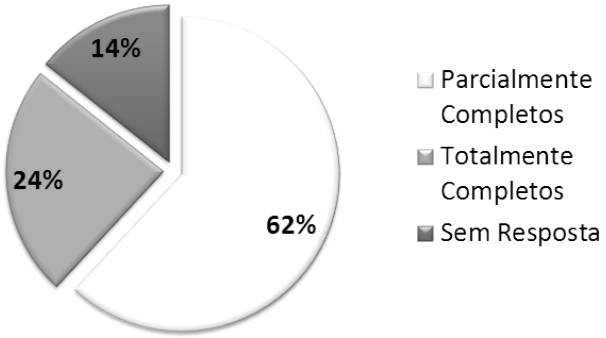
Figura 6.19: Suficiência das funcionalidades e recursos pedagógicos oferecidos.



Fonte: Próprio autor.

Já na Figura 6.20 está representado o resultado para a seguinte questão: "Com relação as suas limitações, as funcionalidades e recursos pedagógicos oferecidos são". Observa-se que 62% afirmaram que estes recursos são parcialmente completos, enquanto 24% afirmaram que estes recursos são totalmente completos, tendo 14% dos alunos não respondido esta questão.

Figura 6.20: Nível de limitação das funcionalidades e recursos pedagógicos oferecidos.

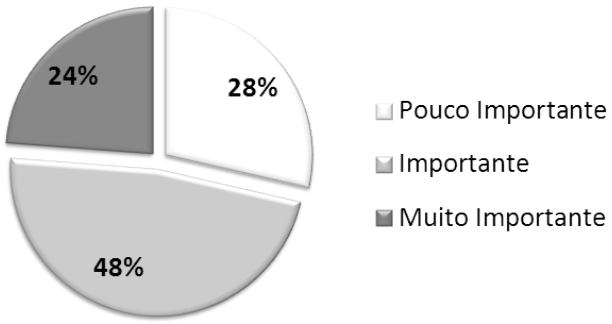


Fonte: Próprio autor.

Quando questionados “Com relação a utilização do AIIP para auxílio do aprendizado de alunos iniciantes em programação, você:” e “Com relação a utilização do AIIP para auxílio nas tarefas de ensino do professor, você:”, 100% dos os alunos afirmaram concordar totalmente. Estas 2 questões e suas consequentes aprovações unânimes serviram para responder parcialmente o problema de pesquisa proposto, sendo, desta forma, possível acreditar que o ambiente aqui proposto pode auxiliar tanto no aprendizado de alunos iniciantes em programação, assim como professores em suas tarefas.

Fazendo uma análise acerca dos ambientes do AIIP (Teórico, Prático, Estatísticas e Gerenciamento), têm-se na Figura 6.21 o resultado para a questão: “Para você, qual o nível de importância do Ambiente Teórico do AIIP?”. Como é possível observar o resultado foi dividido, mas ainda assim é perceptível a aprovação, já que 48% dos alunos consideraram o Ambiente Teórico importante, 28% consideram pouco importante, enquanto 24% consideraram muito importante.

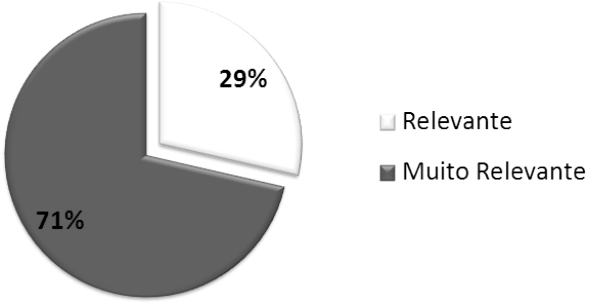
Figura 6.21: Nível de importância do Ambiente Teórico do AIIP.



Fonte: Próprio autor.

O Ambiente de Estatísticas, por sua vez, obteve uma melhor aprovação dos alunos. Como é possível observar na Figura 6.22, 71% dos alunos afirmaram que este Ambiente é muito relevante, enquanto 29% afirmaram que é relevante.

Figura 6.22: Nível de relevância do Ambiente de Estatísticas do AIIP.



Fonte: Próprio autor.

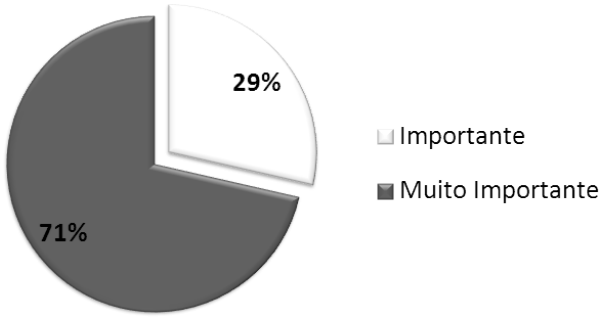
A aprovação do Ambiente de Estatísticas pelos alunos pode ser explicada pelo incentivo

do AIIP à resolução de problemas, uma vez que os alunos tem a sua disposição um recurso onde os mesmos podem visualizar seus desempenhos, no que se refere aos problemas resolvidos, erros cometidos, avaliações, dentre outros fatores, criando assim uma expectativa e sensação de competição para melhorar seu desempenho individual e consequentemente seu aprendizado.

A análise do resultado das questões seguintes é interessante pelo fato das mesmas tratarem sobre o Ambiente de Gerenciamento do AIIP. Embora este Ambiente se destine ao professor, o mesmo também foi apresentado aos alunos para que estes pudessem avaliar as funcionalidades e recursos deste Ambiente, imaginando-se no papel dos professores.

É possível observar na Figura 6.23 que 71% dos alunos consideraram o Ambiente de Gerenciamento muito importante para o AIIP, enquanto 29% consideraram importante, confirmando assim a aprovação deste recurso na ótica dos alunos.

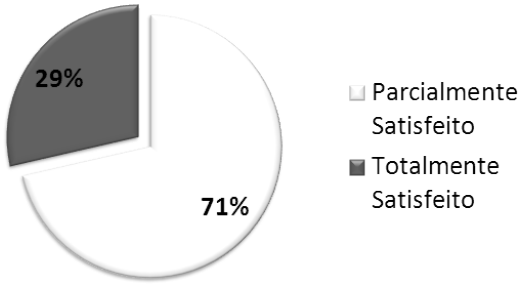
Figura 6.23: Nível de importância do Ambiente de Gerenciamento do AIIP.



Fonte: Próprio autor.

Já na Figura 6.24 observa-se a satisfação dos alunos com relação a flexibilidade no gerenciamento dos recursos do AIIP, onde 71% afirmaram estar parcialmente satisfeitos e 29% afirmaram estar totalmente satisfeitos.

Figura 6.24: Nível de satisfação com a flexibilidade no gerenciamento dos recursos do AIIP.



Fonte: Próprio autor.

Com relação ao gerenciamento dos pesos e pontuações das métricas para avaliação automática dos algoritmos e ao gerenciamento do material instrucional do AIIP, 100% dos alunos

concordaram que estes recursos são completos, desta forma, reafirmando a importância deste ambiente como auxiliador do professor na execução de suas de elaboração, gerenciamento de material instrucional e avaliação de atividades dos alunos.

7 CONSIDERAÇÕES FINAIS E PERSPECTIVAS FUTURAS

É notável que a programação de computadores se configura com umas das disciplinas mais importantes em cursos da área da computação. Da mesma forma, seu aprendizado, para muitos alunos iniciantes, parece ser um processo complexo e cercado de dificuldades.

Através de uma pesquisa bibliográfica realizada sobre o tema, foram encontrados na literatura diversos estudos apresentando diferentes estratégias e abordagens com objetivo de amenizar as dificuldades encontradas pelos alunos e tornar o processo de ensino/aprendizagem de programação menos complexo.

Os STI surgiram como uma alternativa bastante interessante para auxiliar no processo de ensino/aprendizagem, independente do domínio de conhecimento. Entretanto, apesar de muitas pesquisas durante as últimas quatro décadas, desde o surgimento destes sistemas, muitos objetivos ainda não foram atingidos devido a algumas razões, como por exemplo: as limitações de *software* e *hardware*, a complexidade do processo educacional e o custo na construção destes sistemas.

Neste sentido, sabendo das dificuldades existentes para o desenvolvimento de um STI completo, com todas as suas características e funcionalidades desejadas, o AIIP foi desenvolvido. Trata-se de um ambiente apoiado por um assistente inteligente de ensino cujo objetivo é auxiliar alunos e professores no processo de ensino/aprendizagem de programação através da resolução de problemas.

Optou-se pelo desenvolvimento de um ambiente apoiado por um assistente inteligente menos complexo, mas não menos eficiente do que um STI, onde além de auxiliar os alunos, através do oferecimento de dicas e avaliando as suas soluções algorítmicas de forma automática, o mesmo também auxilia o professor em algumas de suas tarefas. Desta forma, a abordagem tradicional de um STI, a qual substitui completamente o papel do professor, deu lugar a uma abordagem colaborativa, onde professor e sistema compartilham informações que possibilitam melhorar a assistência ao aluno.

Além disso, a escolha pela implementação de um assistente inteligente como componente do ambiente se deu pela flexibilidade de manutenção, uma vez que novas funcionalidades para o assistente podem ser desenvolvidas e as funcionalidades já existentes podem ser aprimoradas sem afetar a arquitetura do sistema como um todo.

Com relação as dificuldades encontradas no desenvolvimento deste trabalho, principalmente no aspecto da implementação do ambiente, pode-se destacar a dificuldade na modelagem do assistente inteligente, com relação à suas intervenções junto ao aluno no momento de interação com o ambiente. Apesar da existência e do conhecimento de recursos que possam ser apresentados aos mesmos, determinar o momento em que o aluno necessita de ajuda, ou seja, diagnosticar a necessidade de assistência ao aluno em uma determinada situação, é o ponto mais complicado de ser implementado.

Foram encontradas dificuldades também para realizar a avaliação do ambiente devido a greve de âmbito nacional nas universidades públicas que se estendeu por aproximadamente 4 meses. Assim, não foi possível realizar uma avaliação formal com os professores relacionada à questão do gerenciamento do ambiente. Neste sentido, os próprios alunos exerceram a figura de um professor, respondendo as questões sobre este tópico.

Com relação às contribuições deste trabalho, podemos destacar a metodologia de avaliação do processo de construção das soluções algorítmicas dos alunos, através da análise de 4 fatores: o **Algoritmo** em si, as **Dicas** e os **Refinamentos** solicitados e os **Erros** cometidos, onde para cada um destes fatores propostos, métricas específicas foram criadas com o objetivo de avaliar automaticamente as soluções algorítmicas submetidas pelos alunos.

Outra contribuição deste trabalho é a flexibilidade tanto na forma de avaliação dos alunos, quanto no gerenciamento do conteúdo armazenado no ambiente. As métricas dos fatores e todo o conteúdo instrucional podem ser modificados pelo professor, a depender de diversos aspectos.

A categorização e apresentação das Dicas e dos Refinamentos aos alunos, de acordo com seu nível de conhecimento, além da apresentação de *feedbacks* sobre os erros cometidos durante a construção das soluções algorítmicas em respostas aos problemas, também se configuram como principais contribuições deste trabalho, no sentido de serem estratégias pedagógicas não observadas tão comumente em ambientes de desenvolvimento comerciais e/ou profissionais.

Ou seja, da forma como o AIIP foi idealizado, provendo tanto um ambiente de ensino Teórico (com a apresentação de conceitos e exemplos divididos em unidades de conhecimento), quanto um ambiente de ensino Prático (onde os alunos podem trabalhar de forma independente ou resolver os problemas propostos), além de prover um assistente inteligente para auxiliar os alunos durante a resolução dos problemas e avaliar as soluções dos mesmos de forma automática, de acordo com métricas pré-estabelecidas, é possível crer em sua contribuição, de forma geral, no processo de ensino/aprendizagem de programação.

De posse do resultado da análise dos dados da avaliação realizada com os alunos, apresentada no capítulo anterior, e de conversas informais com alguns professores da área da programação, foi possível concluir que a proposta apresentada neste trabalho pode, até certo ponto, suprir o que se espera no processo de ensino/aprendizagem de programação por um ambiente de aprendizado, respondendo positivamente, mesmo que de forma parcial, devido a falta de uma avaliação formal com professores, o problema proposto na pesquisa.

Outro fato que demonstra a aprovação da proposta aqui apresentada é a aceitação dos trabalhos (artigos) submetidos anteriormente à finalização desta dissertação. No ano de 2011, três trabalhos foram submetidos e todos eles foram aceitos. Na ERBASE - Escola Regional Bahia Alagoas Sergipe, congresso regional realizado anualmente pela SBC, foram aceitos e publicados os trabalhos de Lima et al. (2011) e de Gomes et al. (2011*a*), enquanto no WEI, congresso nacional também realizado pela SBC, o trabalho de Gomes et al. (2011*b*) foi aceito

e publicado, ou seja, reforçando o *feedback* positivo com relação a boa aceitação da proposta apresentada nesta dissertação.

Com relação aos trabalhos futuros resultantes desta dissertação, é possível destacar algumas ideias que poderiam ser implementadas com o intuito de melhorar o ambiente proposto.

Uma vez que esta versão inicial do AIIP está disponível apenas para a plataforma *Windows*, já que a biblioteca externa responsável pelo funcionamento do interpretador possui incompatibilidade com *Linux*, a adaptação desta biblioteca para que o AIIP também possa funcionar na plataforma *Linux* apresenta-se como uma tarefa interessante.

Outra ideia é a migração do ambiente para a *web*, já que o mesmo foi implementado para funcionar apenas em *desktops*. Essa mudança traria muito mais flexibilidade e mobilidade tanto para os professores quanto para os alunos. Seria possível disponibilizar e gerenciar o conteúdo programático (unidades, conceitos, exemplos, etc), além dos problemas e das métricas de avaliação em um único servidor, cabendo aos clientes (alunos e professores) acessarem o ambiente através de um navegador em qualquer computador. A implementação de novos serviços com certeza seria necessária, porém o custo benefício e os resultados futuros poderiam justificar este trabalho.

Um outro possível trabalho futuro, porém um pouco mais complexo, seria a implementação de um processo de avaliação automática dos algoritmos mais completo, ou seja, levando em consideração aspectos qualitativos e não apenas quantitativos. Para tanto, possivelmente, seria necessário a criação de novas métricas e novas regras para que a avaliação pudesse ser realizada da melhor forma possível.

REFERÊNCIAS

Ala-Mutka, K. (2004), Problems in learning and teaching programming - a literature study for developing visualizations in the codewitz-minerva project, Technical report, Institute of Software Systems, Tampere University of Technology, Finland.

Aleven, V. & Koedinger, K. R. (2000), Limitations of student control: do students know when they need help?, *in* 'ITS '00: Proceedings of the 5th International Conference on Intelligent Tutoring Systems', Springer-Verlag, London, UK, pp. 292–303.

Allen, E., Cartwright, R. & Stoler, B. (2002), 'Drjava: a lightweight pedagogic environment for java', *ACM SIGCSE Bulletin* **34**(1), 137–141.

Almeida, E. S., Costa, E. B., Silva, K. S., Paes, R. B., Almeida, A. A. M. & Braga, J. D. H. (2002), Ambap: um ambiente de apoio ao aprendizado de programação, *in* 'WEI 2002: X Workshop Sobre Educação em Computação', Florianópolis, SC, Brasil.

Baecker, R. M. & Sherman, D. (1981), 'Sorting out sorting', Multimedia. URL `http://video.google.com/videoplay?docid=-4110947752111188923&emb=1&hl=pt-BR`, 30-minute 16mm colour sound film. Presented at the ACM SIGGRAPH conference, Dallas, TX, August 3–7.

Basili, V., Caldiera, G. & Rombach, D. H. (1994), The goal question metric approach, *in* J. Marciniak, ed., 'Encyclopedia of Software Engineering', John Wiley & Sons, Inc.

Bravo, C., Marcelino, M. J., Gomes, A., Esteves, M. & Mendes, A. J. (2005), 'Integrating educational tools for collaborative computer programming learning', *Journal of Universal Computer Science* **11**(9), 1505–1517. URL `http://www.jucs.org/jucs_11_9/integrating_educational_tools_for`.

Bravo, C., Redondo, M. A. & Ortega, M. (2004), Aprendizaje en grupo de la programación mediante técnicas de colaboración distribuida en tiempo real, *in* 'Proceedings of V Congreso Interacción Persona Ordenador', Lleida, Spain, pp. 351–357.

Brusilovsky, P. (2001), Webex: learning from examples in a programming course, *in* 'WebNet'2001: Proceedings of World Conference of the WWW and Internet', AACE Press, New York, Orlando, FL, p. 124–129.

Buck, D. & Stucki, D. J. (2001), Jkarelrobot: a case study in supporting levels of cognitive development in the computer science curriculum, *in* 'SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education', ACM, New York, NY, USA, pp. 16–20.

Carbonell, J. R. (1970), 'Ai in cai: an artificial-intelligence approach to computer-assisted instruction', *IEEE Transactions on Man-Machine Systems* **11**(4), 190–202. URL http://dx.doi.org/10.1109/TMMS.1970.299942.

Chou, C.-Y., Chan, T.-W. & Lin, C.-J. (2003), 'Redefining the learning companion: the past, present, and future of educational agents', *Computer & Education* **40**, 255–269. URL http://dl.acm.org/citation.cfm?id=762047.762051.

Cockburn, A. & Williams, L. (2001), *The costs and benefits of pair programming*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, pp. 223–243. URL http://portal.acm.org/citation.cfm?id=377517.377531.

Cooper, S., Dann, W. & Pausch, R. (2003), 'Using animated 3d graphics to prepare novices for cs1', *Computer Science Education* **13**, 3–32.

Crescenzi, P., Demetrescu, C., Finocchi, I. & Petreschi, R. (2000), 'Reversible execution and visualization of programs with leonardo', *Journal of Visual Languages & Computing* **11**(2), 125 – 150.

Deek, F., Kimmel, H. & McHugh, J. (1998), 'Pedagogical changes in the delivery of the first-course in computer science: problem solving, then programming', *Journal of Engineering Education* **87**, 313–320.

Esmin, A. A. A. (1998), Portugol/plus : uma ferramenta de apoio ao ensino da lógica de programação baseado no portugol, *in* 'Anais do IV Congresso Ibero-americano de Informática Educativa', Brasília - Brasil.

Esteves, M. & Mendes, A. J. (2004), A simulation tool to help learning of object oriented programming basics, *in* 'FIE 2004: 34th Annual Frontiers in Education', pp. F4C – 7–12 Vol. 2.

Evaristo, J. & Crespo, S. (2003), *Aprendendo a programar - Programando numa linguagem algorítmica executável (ILA)*, Book Express.

Fernández, A. & Sánchez, J. (2003), 'Cgraphic: educational software for learning the foundations of programming', *Computer Applications in Engineering Education* **11**(4), 167–178.

Freund, S. N. & Roberts, E. S. (1996), 'Thetis: an ansi c programming environment designed for introductory use', *ACM SIGCSE Bulletin* **28**(1), 300–304.

Giraffa, L. M. M. (1999), Uma arquitetura de tutor utilizando estados mentais, PhD thesis, Instituto de Informática, Programa de Pós-Graduação em Computação - Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil.

Giraffa, L., Marczak, S. & Almeida, G. (2003), O ensino de algoritmos e programação mediado por um ambiente na web, *in* 'Anais do Congresso Nacional da Sociedade Brasileira de Computação', Campinas-SP.

Goldman, K. J. (2004), 'An interactive environment for beginning java programmers', *Science of Computer Programming* **53**(1), 3 – 24. Practice and experience with Java in education.

Gomes, C. C. C., Lima, D. H. S., Ribeiro, R. P., Almeida, E. S. & Brito, P. H. S. (2011*a*), Aprendizagem de programação com base na resolução de problemas em um ambiente apoiado por um assistente inteligente, *in* 'XI ERBASE: IX Workshop de Educação e Informática'.

Gomes, C. C. C., Lima, D. H. S., Ribeiro, R. P., Almeida, E. S. & Brito, P. H. S. (2011*b*), Uma proposta para auxiliar alunos e professores no ensino de programação: O ambiente aiip, *in* 'XXXI CSBC: XIX Workshop sobre Educação em Computação'.

Gómez-Albarrán, M. (2005), 'The teaching and learning of programming: a survey of supporting software tools', *The Computer Journal* **48**(2), 130–144.

Haajanen, J., Pesonius, M., Sutinen, E., Tarhio, J., Terasvirta, T. & Vanninen, P. (1997), Animation of user algorithms on the web, *in* 'VL '97: Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97)', IEEE Computer Society, Washington, DC, USA, p. 356.

Han, K., Lee, E. & Lee, Y. (2006), 'The effects of pair programming on achievement and motivated strategies in programming', *Computer & Education* **9**(6), 19–28.

Han, K., Lee, E. & Lee, Y. (2010), 'The impact of a peer-learning agent based on pair programming in a programming course', *IEEE Transactions on Education* **53**(2), 318 –327.

Hmelo-Silver, C. (2004), 'Problem-based learning: What and how do students learn?', *Educational Psychology Review* **16**, 235–266. URL `http://dx.doi.org/10.1023/B:EDPR.0000034022.16470.f3`.

Jenkins, T. (2002), On the difficulty of learning to program, *in* 'Proceedings of 3rd annual conference of the LTSN-ICS', Loughbourgh. URL `http://www.ics.ltsn.ac.uk/pub/conf2002/tjenkins.pdf`.

Júnior, J. C. R. P. & Rapkiewicz, C. E. (2006), O processo de ensino-aprendizagem de fundamentos de programação: uma visão crítica da pesquisa no brasil, *in* 'WEI 2006: XIII Workshop sobre Educação em Computação', Campo Grande, MS, Brasil.

Kelly, J., Mooney, A., Ghent, J., Gaughran, P., Dunne, S. & Bergin, S. (2004), An overview of the integration of problem-based learning into an existing computer science programming module, *in* 'PBL Internacional Conference - Pleasure by Learning', Mexico.

Kinnunen, P. & Malmi, L. (2002), Problem based learning in introductory programming - does it scale up?, *in* 'II Finnish/Baltic Sea Conference of Computer Science Education', Department of Computer Science, University of Joensuu, pp. 38–42.

Kölling, M. & Rosenberg, J. (1996), 'Blue - a language for teaching object-oriented programming', *ACM SIGCSE Bulletin* **28**(1), 190–194.

Kölling, M., Quig, B., Patterson, A. & Rosenberg, J. (2003), 'The bluej system and its pedagogy', *Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology* **13**(4), 249–268.

Lahtinen, S., Sutinen, E. & Tarhio, J. (1998), 'Automated animation of algorithms with eliot', *Journal of Visual Languages & Computing* **9**, 337–349.

Levy, R., Ben-Ari, M. & Uronen, P. (2003), 'The jeliot 2000 program animation system', *Computer & Education* **40**, 1–15.

Lima, D. H. S., Gomes, C. C. C., Ribeiro, R. P., Brito, P. H. S. & Almeida, E. S. (2011), Uma proposta de arquitetura para um ambiente inteligente de aprendizagem de programação baseado na resolução de problemas, *in* 'XI ERBASE: Workshop de Trabalhos de Iniciação Científica e Graduação'.

Mayer, R. E., Johnson, W. L., Shaw, E. & Sandhu, S. (2006), 'Constructing computer-based tutors that are socially sensitive: politeness in educational software', *International Journal of Human-Computer Studies* **64**(1), 36–42.

McDowell, C., Werner, L., Bullock, H. E. & Fernald, J. (2006), 'Pair programming improves student retention, confidence, and program quality', *Communications of the ACM* **49**(8), 90–95.

Mendes, A. J. N. & Gomes, A. J. (2000), Suporte à aprendizagem da programação com o ambiente sicas, *in* 'V Congresso Ibero-Americano de Informática Educativa', Viña del Mar, Chile.

Ministério da Educação, M. (1999), *Diretrizes curriculares de cursos da área de computação e informática*, Comissão de especialistas de ensino de computação e informática - CEEInf, Brasília, DF, Brasil.

Muñoz-Merino, P. J. & Kloos, C. D. (2009), 'A software player for providing hints in problem-based learning according to a new specification', *Computer Applications in Engineering Education* **17**(3), 272–284.

Naps, T. L., Eagan, J. R. & Norton, L. L. (2000), JhavÉ - an environment to actively engage students in web-based algorithm visualizations, *in* 'SIGCSE '00: Proceedings of the

thirty-first SIGCSE technical symposium on Computer science education', ACM, New York, NY, USA, pp. 109–113.

Nobre, I. A. M. & Menezes, C. S. (2002), Suporte à cooperação em um ambiente de aprendizagem para programação (samba), *in* 'SBIE'2002: XIII Simpósio Brasileiro de Informática na Educação', São Leopoldo-RS, Brasil.

Pattis, R. E. (1981), *Karel the robot: a gentle introduction to the art of programming*, John Wiley and Sons, Inc, New York, USA.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. & Paterson, J. (2007), 'A survey of literature on the teaching of introductory programming', *ACM SIGCSE Bulletin* **39**(4), 204–223.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F. & Simmons, R. (1986), 'Conditions of learning in novice programmers', *Journal of Educational Computing Research* **2**(1), 37 – 55.

Petry, P. G. & Rosatelli, M. C. (2006), Ensino e aprendizagem de algoritmos com o algolc, *in* 'SBIE 2006: XVII Simpósio Brasileiro de Informática na Educação', Brasília, DF, Brasil.

Pillay, N. (2003), 'Developing intelligent programming tutors for novice programmers', *ACM SIGCSE Bulletin* **35**(2), 78–82.

Pinto, S. C. C. S. (1995), M-assiste: Um meta-assistente adaptativo para suporte à navegação em documentos hipermídia, Master's thesis, COPPE/UFRJ - Universidade Federal do Rio de Janeiro.

Raabe, A. L. A. (2005), Uma proposta de arquitetura de sistema tutor inteligente baseada na teoria das experiências de aprendizagem mediadas, PhD thesis, Programa de Pós-Graduação em Informática na Educação - Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil.

Redondo, M. A., Bravo, C., Ortega, M. & Verdejo, F. (2002), Planedit: an adaptive problem solving tool for design, *in* 'AH '02: Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems', Springer-Verlag, London, UK, pp. 560–563.

Robins, A., Rountree, J. & Rountree, N. (2003), 'Learning and teaching programming: a review and discussion', *Computer Science Education* **13**, 137–172. URL `http://www.informaworld.com/10.1076/csed.13.2.137.14200`.

Rocha, H. V. (1991), Representações computacionais auxiliares ao entedimento de conceitos de programação, PhD thesis, Faculdade de Engenharia Elétrica e de Computação,

Programa de Pós-Graduação em Engenharia Elétrica - Universidade Estadual de Campinas, Campinas, SP, Brasil.

Rössling, G. & Freisleben, B. (2000), Experiences in using animations in introductory computer science lectures, *in* 'SIGCSE '00: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education', ACM, New York, NY, USA, pp. 134–138.

Satratzemi, M., Dagdilelis, V. & Evagelidis, G. (2001), 'A system for program visualization and problem-solving path assessment of novice programmers', *ACM SIGCSE Bulletin* **33**(3), 137–140.

Schneider, G. M. (1978), The introductory programming course in computer science: ten principles, *in* 'SIGCSE '78: Papers of the SIGCSE/CSA technical symposium on Computer science education', ACM, New York, NY, USA, pp. 107–114.

Shaw, M. & Garlan, D. (1996), *Software architecture: perspectives on an emerging discipline*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Sousa, S. O. (2011), Aprendizagem baseada em problemas (pbl - problem based learning): Estratégia para o ensino e aprendizagem de algoritmos e conteúdos computacionais, Master's thesis, UNESP - Universidade Estadual Paulista.

Souza, C. M. (2009), 'Visualg - ferramenta de apoio ao ensino de programação', *Revista Eletrônica TECCEN* **2**(2), 1–9.

Spohrer, J. C., Soloway, E. & Pope, E. (1985), 'A goal/plan analysis of buggy pascal programs', *Human-Computer Interaction* **1**(2), 163–207.

Vanlehn, K. (2006), 'The behavior of tutoring systems', *International Journal of Artificial Intelligence in Education* **16**(3), 227–265.

Weber, G. (1996), 'Individual selection of examples in an intelligent learning environment', *Journal of Artificial Intelligence in Education* **7**, 3–31.

Weber, G. & Brusilovsky, P. (2001), 'Elm-art: an adaptive versatile system for web-based instruction', *International Journal of Artificial Intelligence in Education, Special Issue on Adaptive and Intelligent Web-based Educational Systems* **12**, 351–384.

Wenger, E. (1987), *Artificial intelligence and tutoring systems: computational and cognitive approaches to the communication of knowledge*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Winslow, L. E. (1996), 'Programming pedagogy - a psychological overview', *ACM SIGCSE Bulletin* **28**(3), 17–22.

Yacef, K. (2002), Intelligent teaching assistant systems, *in* 'ICCE'02: International Conference on Computers in Education', Auckland, New Zealand.

Ásrún Matthíasdóttir (2006), How to teach programming languages to novice students? lecturing or not?, *in* 'CompSysTech'06: Proceedings of the International Conference on Computer Systems and Technologies', Bulgaria, pp. 1–7.

A QUESTIONÁRIO SOBRE ASPECTOS RELEVANTES NA AVALIAÇÃO DE ALGORITMOS

Para cada item abaixo, indicar o nível de relevância do mesmo no momento da avaliação de um algoritmo apresentado por um aluno, em resposta a um problema específico.
Obs.: Considere que o algoritmo apresenta um resultado igual ou satisfatório ao resultado esperado.

Quantidade de linhas de código utilizada:			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Quantidade de variáveis declaradas:			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Quantidade de variáveis declaradas, não utilizadas:			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Coerência dos nomes das variáveis: (As variáveis apresentam nomes coerentes com o próposito que se destinam?)			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Quantidade de instruções utilizadas: (Considere como instruções todas as expressões aritméticas, relacionais e lógicas, os comandos de atribuição, além dos comandos de entrada e saída)			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Quantidade de estruturas condicionais utilizadas: (Levando em consideração que o problema requer a implementação deste tipo de estrutura)			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Quantidade de estruturas de repetição utilizadas: (Levando em consideração que o problema requer a implementação deste tipo de estrutura)			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Quantidade de funções ou procedimentos utilizados: (Levando em consideração que o problema requer a implementação deste tipo de estrutura)			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Organização visual do código (Identação)			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Quantidade de erros cometidos: (Sintáticos e Semânticos)			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Além dos itens listados acima, quais outros fatores você considera importantes/relevantes no momento de avaliar o algoritmo de um aluno em resposta a um problema específico?			
_____ _____ _____ _____ _____			
Disciplina(s) que já lecionou:			
( ) Laboratório de Programação	( ) Introdução à Programação	( ) Programação 1	( ) Programação 2

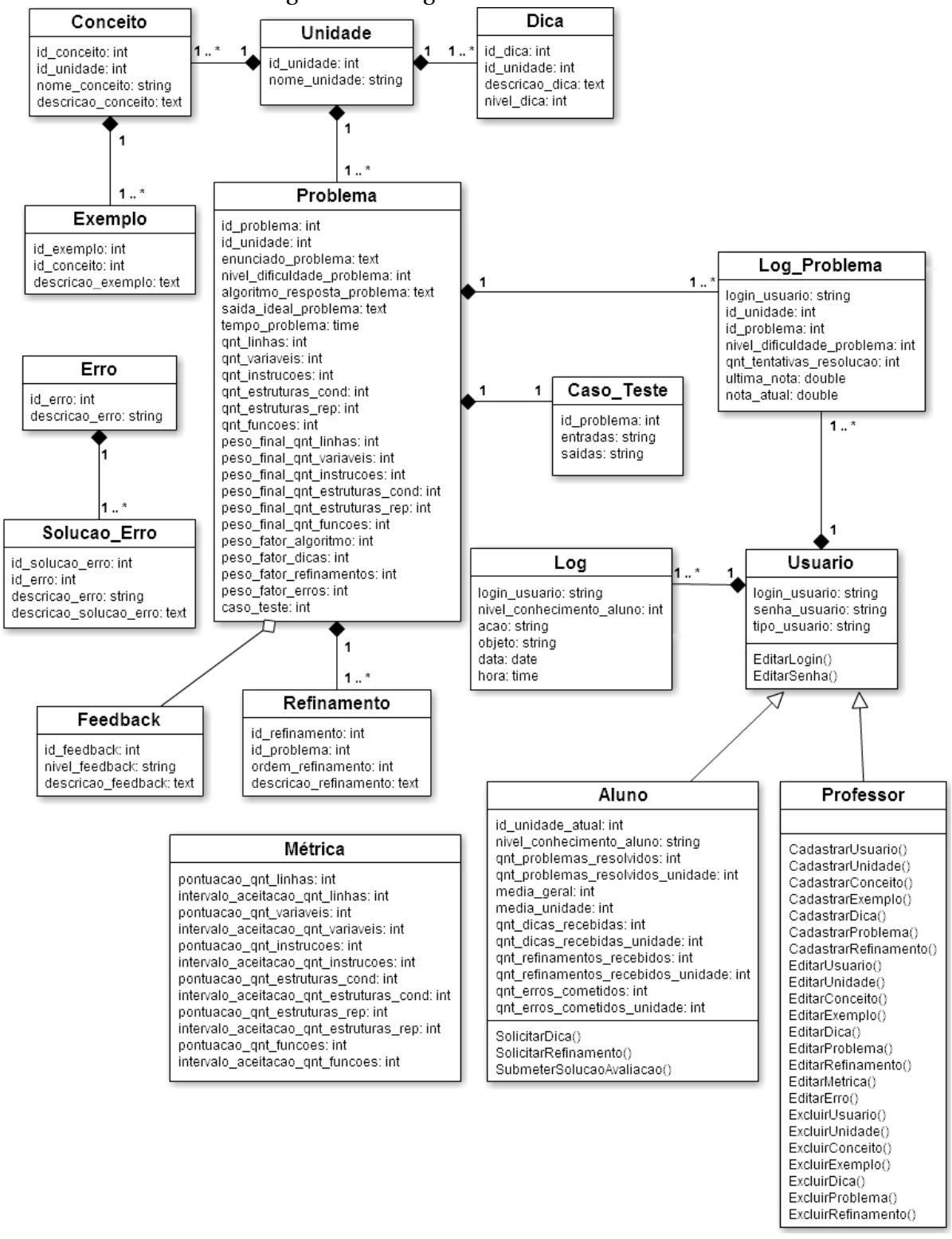
B QUESTIONÁRIO SOBRE SATISFAÇÃO DE USO DO AIIP

Você costuma utilizar ou já utilizou alguma IDE comercial/profissional para desenvolver seus algoritmos?			
( ) Sim		( ) Não	
Se sim, qual ou quais?			
_____			
_____			
Das linguagens listadas abaixo, assinale as que você, pelo menos, já ouviu falar.			
( ) Logo	( ) Eiffel	( ) Scratch	( ) Pascal
( ) C	( ) C++	( ) Java	( ) Phyton
Você costuma desenvolver ou já desenvolveu algoritmos em alguma(s) das linguagens listadas na questão anterior?			
( ) Sim		( ) Não	
Se sim, qual ou quais linguagens?			
_____			
_____			
Em comparação com alguma IDE já utilizada por você, com relação aos seus recursos e funcionalidades de forma geral, o AIIP pode ser considerado:			
( ) Muito Inferior	( ) Inferior	( ) Superior	( ) Muito Superior
Na sua opinião, a utilização dos recursos de edição, compilação e execução (normal e passo-a-passo) dos algoritmos é:			
( ) Muito Dificil	( ) Dificil	( ) Fácil	( ) Muito Fácil
Na sua opinião, qual a relevância do simulador de execução passo-a-passo do código?			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Com relação a limitação, os recursos e funcionalidades do ambiente, em um âmbito geral, são:			
( ) Muito Limitados	( ) Limitados	( ) Completos	( ) Muito Completos
De forma geral, os recursos e funcionalidades do ambiente são:			
( ) Muito Bem Organizados	( ) Bem Organizados	( ) Mal Organizados	( ) Muito Mal Organizados
Na sua opinião, qual o nível de dificuldade de entendimento da sintaxe da linguagem?			
( ) Muito Fácil	( ) Fácil	( ) Difícil	( ) Muito Difícil
Com relação a simplicidade, a linguagem apresenta-se:			
( ) Muito Simples	( ) Simples	( ) Complexa	( ) Muito Complexa
Como você classificaria a linguagem, com relação à sua limitação?			
( ) Totalmente Limitada	( ) Limitada	( ) Completa	( ) Totalmente Completa
Com relação a utilização da resolução de problemas como estratégia pedagógica, você:			
( ) Discorda Totalmente	( ) Discorda Parcialmente	( ) Concorda Parcialmente	( ) Concorda Totalmente
Como você avalia a importância do Assistente Inteligente para o ambiente?			
( ) Sem Importância	( ) Pouco Importante	( ) Importante	( ) Muito Importante
Qual seu grau de satisfação com a metodologia de avaliação automática utilizada no AIIP?			
( ) Totalmente Insatisfeito	( ) Parcialmente Instisfeito	( ) Parcialmente Satisfeito	( ) Totalmente Satisfeito

Qual seu nível de satisfação com os feedbacks recebidos sobre os erros cometidos?			
( ) Totalmente Insatisfeito	( ) Parcialmente Instisfeito	( ) Parcialmente Satisfeito	( ) Totalmente Satisfeito
Para você, os recursos de ajuda e feedbacks oferecidos são:			
( ) Totalmente Insuficientes	( ) Parcialmente Insuficientes	( ) Parcialmente Suficientes	( ) Totalmente Suficientes
Para você, as funcionalidades e recursos pedagógicos oferecidos são:			
( ) Totalmente Insuficientes	( ) Parcialmente Insuficientes	( ) Parcialmente Suficientes	( ) Totalmente Suficientes
Com relação às suas limitações, as funcionalidades e recursos pedagógicos oferecidos são:			
( ) Totalmente Limitados	( ) Parcialmente Limitados	( ) Parcialmente Completos	( ) Totalmente Completos
Com relação a utilização do ambiente para auxílio do aprendizado de alunos iniciantes em programação, você:			
( ) Discorda Totalmente	( ) Discorda Parcialmente	( ) Concorda Parcialmente	( ) Concorda Totalmente
Com relação a utilização do ambiente para auxílio nas tarefas de ensino do professor, você:			
( ) Discorda Totalmente	( ) Discorda Parcialmente	( ) Concorda Parcialmente	( ) Concorda Totalmente
Para você, qual o nível de importância do Ambiente Teórico de ensino do AIIP?			
( ) Sem Importância	( ) Pouco Importante	( ) Importante	( ) Muito Importante
Na sua opinião, o Ambiente de Estatísticas apresenta-se:			
( ) Sem Relevância	( ) Pouco Relevante	( ) Relevante	( ) Muito Relevante
Na sua opinião, qual a importância do Ambiente de Gerenciamento para o AIIP?			
( ) Sem Importância	( ) Pouco Importante	( ) Importante	( ) Muito Importante
Qual seu índice de satisfação com a flexibilidade de gerenciamento dos recursos do ambiente?			
( ) Totalmente Insatisfeito	( ) Parcialmente Instisfeito	( ) Parcialmente Satisfeito	( ) Totalmente Satisfeito
Para você, o gerenciamento dos pesos e pontuações das métricas para avaliação automática dos algoritmos é:			
( ) Muito Limitado	( ) Limitado	( ) Completo	( ) Muito Completo
Para você, o gerenciamento do material instrucional do ambiente é:			
( ) Muito Limitado	( ) Limitado	( ) Completo	( ) Muito Completo

C DETALHAMENTO DAS CLASSES DO AIIP

Figura C.1: Diagrama de Classes do AIIP.



Fonte: Próprio autor.

Classe Unidade

A classe Unidade representa as unidades de conhecimento do ambiente de ensino teórico do AIIP e possui os seguintes atributos:

- `id_unidade`: representa o identificador único de cada unidade, armazenado como `int`.
- `nome_unidade`: representa o nome da unidade, armazenado como uma `string`.

Classe Conceito

A classe Conceito representa os conceitos relacionados às unidades de conhecimento do ambiente teórico do AIIP e possui os seguintes atributos:

- `id_conceito`: representa o identificador único de cada conceito, armazenado como `int`.
- `id_unidade`: representa o identificador único de cada unidade a qual o conceito está ligado, armazenado como `int`.
- `nome_conceito`: representa o nome do conceito, armazenado como uma `string`.
- `descricao_conceito`: representa a descrição textual do conceito, armazenado como `text`.

Classe Exemplo

A classe Exemplo representa os exemplos relacionados aos conceitos do ambiente teórico do AIIP e possui os seguintes atributos:

- `id_exemplo`: representa o identificador único de cada exemplo, armazenado como `int`.
- `id_conceito`: representa o identificador único de cada conceito a qual o exemplo está ligado, armazenado como `int`.
- `descricao_exemplo`: representa a descrição textual do exemplo, armazenado como `text`.

Classe Problema

A classe Problema representa os problemas relacionados às unidades de conhecimento do ambiente teórico do AIIP e possui os seguintes atributos:

- `id_problema`: representa o identificador único de cada problema, armazenado como `int`.
- `id_unidade`: representa o identificador único de cada unidade a qual o problema está relacionado, armazenado como `int`.
- `enunciado_problema`: representa o enunciado do problema, armazenado como `text`.
- `nivel_dificuldade_problema`: representa o nível de dificuldade do problema com relação à unidade de conhecimento a qual ele está associado, armazenado como `int`.
- `algoritmo_resposta_problema`: representa a solução algorítmica ideal para resolver o problema, de acordo com o professor, armazenado como `text`.
- `saida_ideal_problema`: representa a saída que o programa deve gerar, armazenado como `text`.
- `tempo_problema`: representa o tempo médio de resolução do problema, armazenado como `time`.
- `qnt_linhas`: representa a quantidade de linhas do algoritmo resposta cadastrado, armazenado como `int`.
- `qnt_variaveis`: representa a quantidade de variáveis declaradas do algoritmo resposta cadastrado, armazenado como `int`.
- `qnt_instrucoes`: representa a quantidade de instruções do algoritmo resposta cadastrado, armazenado como `int`.
- `qnt_estruturas_cond`: representa a quantidade de estruturas condicionais do algoritmo resposta cadastrado, armazenado como `int`.
- `qnt_estruturas_rep`: representa a quantidade de estruturas de repetição do algoritmo resposta cadastrado, armazenado como `int`.
- `qnt_funcoes`: representa a quantidade de funções do algoritmo resposta cadastrado, armazenado como `int`.
- `peso_final_qnt_linhas`: representa o peso da métrica “quantidade de linhas de código” para o problema, armazenado como `int`.
- `peso_final_qnt_variaveis`: representa o peso da métrica “quantidade de variáveis declaradas” para o problema, armazenado como `int`.
- `peso_final_qnt_instrucoes`: representa o peso da métrica “quantidade de instruções” para o problema, armazenado como `int`.

- `peso_final_qnt_estruturas_cond`: representa o peso da métrica "quantidade de estruturas condicionais" para o problema, armazenado como `int`.
- `peso_final_qnt_estruturas_rep`: representa o peso da métrica "quantidade de estruturas de repetição" para o problema, armazenado como `int`.
- `peso_final_qnt_funcoes`: representa o peso para a métrica "quantidade de funções" para o problema, armazenado como `int`.
- `peso_fator_algoritmo`: representa o peso do fator "Algoritmo" para o problema, armazenado como `int`.
- `peso_fator_dicas`: representa o peso do fator "Dicas" para o problema, armazenado como `int`.
- `peso_fator_refinamentos`: representa o peso do fator "Refinamentos" para o problema, armazenado como `int`.
- `peso_fator_erros`: representa o peso do fator "Erros" para o problema, armazenado como `int`.

Classe Dica

A classe Dica representa as dicas relacionadas às unidades de conhecimento do ambiente teórico do AIIP e possui os seguintes atributos:

- `id_dica`: representa o identificador único de cada dica, armazenado como `int`.
- `id_unidade`: representa o identificador único de cada unidade a qual a dica está relacionada, armazenado como `int`.
- `descricao_dica`: representa a descrição textual da dica, armazenado como `text`.
- `nivel_dica`: representa o nível da dica, armazenado como `int`.

Classe Refinamento

A classe Refinamento representa os refinamentos relacionados aos problemas do ambiente teórico do AIIP e possui os seguintes atributos:

- `id_refinamento`; representa o identificador único de cada refinamento, armazenado como `int`.
- `id_problema`: representa o identificador único de cada problema a qual o refinamento está relacionado, armazenado como `int`.

- `ordem_refinamento`: representa a ordem de apresentação do refinamento, armazenado como `int`.
- `descricao_refinamento`: representa a descrição textual do refinamento, armazenado como `text`.

Classe Erro

A classe Erro representa os erros reconhecidos pelo AIIP, e possui os seguintes atributos:

- `id_erro`: representa o identificador único de cada erro, armazenado como `int`.
- `descricao_erro`: representa a descrição de cada erro, armazenado como `string`.

Classe Solucao_Erro

A classe Solucao_Erro representa as causas e possíveis soluções para os erros reconhecidos pelo AIIP, e possui os seguintes atributos:

- `id_solucao_erro`: representa o identificador único de cada solução/causa do erro encontrado, armazenado como `int`.
- `id_erro`: representa o identificador único de cada erro, armazenado como `int`.
- `descricao_erro`: representa a descrição de cada erro, armazenado como `string`.
- `descricao_solucao_erro`: representa a descrição de cada solução/causa do erro encontrado, armazenado como `string`.

Classe Caso_Teste

A classe Caso_Teste representa os casos de teste dos problemas armazenados no AIIP e possui os seguintes atributos:

- `id_problema`: representa o identificador único de cada problema, armazenado como `int`.
- `entradas`: representa as entradas do problema para que o mesmo apresente uma resposta satisfatória, armazenado como `string`.
- `saidas`: representa as saídas do problema, de acordo com as entradas armazenadas, para que o mesmo apresente uma resposta satisfatória, armazenado como `string`.

Classe *Feedback*

A classe *Feedback* representa os *feedbacks* as métricas utilizadas para avaliação automática dos algoritmos desenvolvidos em resposta aos problemas armazenados no AIIP e possui os seguintes atributos:

- `id_feedback`: representa o identificador único de cada *feedback*, armazenado como `int`.
- `nivel_feedback`: representa o nível do *feedback*, armazenado como `int`.
- `descricao_feedback`: representa a descrição textual do *feedback*, armazenado como `text`.

Classe Usuario

A classe Usuario representa os usuários que possuem acesso ao AIIP e possui os seguintes atributos e métodos:

- `login_usuario`: representa o login do usuário para acesso ao AIIP.
- `senha_usuario`: representa a senha do usuário para acesso ao AIIP.
- `tipo_usuario`: representa o tipo de usuário e seu respectivo privilégio.
- `EditarLogin()`: responsável pela alteração do login do usuário para acesso ao AIIP.
- `EditarSenha()`: responsável pela alteração da senha do usuário para acesso ao AIIP.

Classe Aluno

A classe Aluno é uma subclasse da classe Usuario, herdando seus atributos e métodos, e possui os seguintes atributos e métodos adicionais:

- `id_unidade_atual`: representa o identificador único da unidade a qual o aluno está estudando e/ou resolvendo problemas, armazenado como `int`.
- `nivel_conhecimento_aluno`: representa o nível do conhecimento do aluno, com relação à unidade a qual o aluno está estudando e/ou resolvendo problemas, armazenado como `int`.
- `qnt_problemas_resolvidos`: representa a quantidade total de problemas já resolvidos pelo aluno, armazenado como `int`.

- `qnt_problemas_resolvidos_unidade`: representa a quantidade de problemas já resolvidos pelo aluno, como relação à unidade a qual o aluno está estudando e/ou resolvendo problemas, armazenado como `int`.
- `media_geral`: representa a média geral das notas dos problemas já resolvidos pelo aluno, armazenado como `int`.
- `media_unidade`: representa a média das notas dos problemas já resolvidos pelo aluno, com relação à unidade a qual o aluno está estudando e/ou resolvendo problemas, armazenado como `int`.
- `qnt_dicas_recebidas`: representa a quantidade total de dicas recebidas pelo aluno, armazenado como `int`.
- `qnt_dicas_recebidas_unidade`: representa a quantidade total de dicas recebidas pelo aluno, referente à unidade a qual o aluno está estudando e/ou resolvendo problemas, armazenado como `int`.
- `qnt_refinamentos_recebidos`: representa a quantidade total de refinamentos recebidos pelo aluno, armazenado como `int`.
- `qnt_refinamentos_recebidos_unidade`: representa a quantidade total de refinamentos recebidos pelo aluno, referente aos problemas da unidade a qual o aluno está estudando e/ou resolvendo problemas, armazenado como `int`.
- `qnt_erros_cometidos`: representa a quantidade total de erros cometidos pelo aluno, armazenado como `int`.
- `qnt_erros_cometidos_unidade`: representa a quantidade total de erros cometidos pelo aluno, referente aos problemas da unidade a qual o aluno está estudando e/ou resolvendo problemas, armazenado como `int`.
- `SolicitarDica()`: responsável pela solicitação de uma dica no momento da resolução de um problema.
- `SolicitarRefinamento()`: responsável pela solicitação de um refinamento no momento da resolução de um problema.
- `SubmeterSolucaoAvaliacao()`: responsável pela submissão da solução algorítmica para avaliação em resposta a um problema.

Classe Professor

A classe Professor é uma subclasse da classe Usuario, herdando seus atributos e métodos, e possui os seguintes métodos adicionais:

- `CadastrarUsuario()`: responsável pelo cadastro de um novo usuário na base de dados.
- `CadastrarUnidade()`: responsável pelo cadastro de uma nova unidade na base de dados.
- `CadastrarConceito()`: responsável pelo cadastro de um novo conceito na base de dados.
- `CadastrarExemplo()`: responsável pelo cadastro de um novo exemplo na base de dados.
- `CadastrarDica()`: responsável pelo cadastro de uma nova dica na base de dados.
- `CadastrarProblema()`: responsável pelo cadastro de um novo problema na base de dados.
- `CadastrarRefinamento()`: responsável pelo cadastro de um novo refinamento na base de dados.
- `CadastrarFeedback()`: responsável pelo cadastro de um novo *feedback* na base de dados.
- `EditarUsuario()`: responsável pela edição de um usuário na base de dados.
- `EditarUnidade()`: responsável pela edição de uma unidade na base de dados.
- `EditarConceito()`: responsável pela edição de um conceito na base de dados.
- `EditarExemplo()`: responsável pela edição de um exemplo na base de dados.
- `EditarDica()`: responsável pela edição de uma dica na base de dados.
- `EditarProblema()`: responsável pela edição de um problema na base de dados.
- `EditarRefinamento()`: responsável pela edição de um refinamento na base de dados.
- `EditarFeedback()`: responsável pela edição de um *feedback* na base de dados.
- `EditarMetrica()`: responsável pela edição de uma métrica na base de dados.
- `ExcluirUsuario()`: responsável pela exclusão de um usuário da base de dados.
- `ExcluirUnidade()`: responsável pela exclusão de uma unidade da base de dados.
- `ExcluirConceito()`: responsável pela exclusão de um conceito da base de dados.
- `ExcluirExemplo()`: responsável pela exclusão de um exemplo da base de dados.

- `ExcluirDica()`: responsável pela exclusão de uma dica da base de dados.
- `ExcluirProblema()`: responsável pela exclusão de um problema da base de dados.
- `ExcluirRefinamento()`: responsável pela exclusão de um refinamento da base de dados.
- `ExcluirFeedback()`: responsável pela exclusão de um *feedback* da base de dados.

Classe Metrica

A classe Metrica representa as métricas utilizadas para avaliação automática dos algoritmos desenvolvidos em resposta aos problemas armazenados no AIIP e possui os seguintes atributos:

- `pontuacao_qnt_linhas`: representa a pontuação correspondente à ocorrência de cada unidade da métrica "quandidade de linhas" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `intervalo_aceitacao_qnt_linhas`: representa o percentual aceitável (mínimo e máximo) do somatório da pontuação da métrica "quandidade de linhas" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `pontuacao_qnt_variaveis`: representa a pontuação correspondente à ocorrência de cada unidade da métrica "quandidade de variaveis" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `intervalo_aceitacao_qnt_variaveis`: representa o percentual aceitável (mínimo e máximo) do somatório da pontuação da métrica "quandidade de variaveis" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `pontuacao_qnt_instrucoes`: representa a pontuação correspondente à ocorrência de cada unidade da métrica "quandidade de instruções" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `intervalo_aceitacao_qnt_instrucoes`: representa o percentual aceitável (mínimo e máximo) do somatório da pontuação da métrica "quandidade de instruções" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `pontuacao_qnt_estruturas_cond`: representa a pontuação correspondente à ocorrência de cada unidade da métrica "quandidade de estruturas condicionais" nos algoritmos submetidos para avaliação, armazenado como `int`.

- `intervalo_aceitacao_qnt_estruturas_cond`: representa o percentual aceitável (mínimo e máximo) do somatório da pontuação da métrica "quandidade de estruturas condicionais" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `pontuacao_qnt_estruturas_rep`: representa a pontuação correspondente à ocorrência de cada unidade da métrica "quandidade de estruturas de repetição" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `intervalo_aceitacao_qnt_estruturas_rep`: representa o percentual aceitável (mínimo e máximo) do somatório da pontuação da métrica "quandidade de estruturas de repetição" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `pontuacao_qnt_funcoes`: representa a pontuação correspondente à ocorrência de cada unidade da métrica "quandidade de funções" nos algoritmos submetidos para avaliação, armazenado como `int`.
- `intervalo_aceitacao_qnt_funcoes`: representa o percentual aceitável (mínimo e máximo) do somatório da pontuação da métrica "quandidade de funções" nos algoritmos submetidos para avaliação, armazenado como `int`.

Classe Log

A classe Log representa os registros das ações realizadas pelos usuário no AIIP, e possui os seguintes atributos:

- `login_usuario`: representa o login do usuário para acesso ao AIIP.
- `nivel_conhecimento_aluno`: representa o nível do conhecimento do aluno, com relação à unidade a qual o aluno está estudando e/ou resolvendo problemas, armazenado como `int`.
- `acao`: representa a ação realizada pelo usuário no AIIP, armazenada como `string`.
- `objeto`: representa o objeto que sofreu a ação (Problema, Erro, etc), armazenado como `string`.
- `data`: representa a data da ocorrência do registro, armazenado como `date`.
- `hora`: representa a hora da ocorrência do registro, armazenado como `time`.

Classe Log_Problema

A classe Log_Problema representa os registros das ações realizadas pelos usuário no AIIP, com relação à resolução dos problemas armazenados, e possui os seguintes atributos:

- `login_usuario`: representa o login do usuário para acesso ao AIIP.
- `id_unidade`: representa o identificador único de cada unidade a qual o problema está relacionado, armazenado como `int`.
- `id_problema`: representa o identificador único de cada problema, armazenado como `int`.
- `nivel_dificuldade_problema`: representa o nível de dificuldade do problema com relação à unidade de conhecimento a qual ele está associado, armazenado como `int`.
- `qnt_tentativas_resolucao`: representa a quantidade de tentativas de resolução do problema pelo usuário, armazenado como `int`.
- `ultima_nota`: representa a última nota recebida pelo usuário para o problema, armazenado como `double`.
- `nota_atual`: representa a nota atual do usuário para o problema, armazenado como `double`.

Este trabalho foi redigido em LaTeX utilizando uma modificação do estilo IC-UFAL. As referências bibliográficas foram preparadas no JabRef e administradas pelo BibTeX com o estilo LaCCAN. O texto utiliza fonte Fourier-GUT*enberg* e os elementos matemáticos a família tipográfica Euler Virtual Math, ambas em corpo de 12 pontos.