Uma Proposta para Auxiliar Alunos e Professores no Ensino de Programação: O Ambiente AIIP

Cledson Calaça Cavalcante Gomes[1], **David Henrique de Souza Lima**[2], **Raphael Pereira Ribeiro**[2], **Eliana Silva de Almeida**[2], **Patrick Henrique da Silva Brito**[2]

[1]Mestrado em Modelagem Computacional do Conhecimento
Universidade Federal de Alagoas (UFAL)
CEP: 57072-970 – Maceió – Alagoas – Brasil

[2]Instituto de Computação – Universidade Federal de Alagoas (UFAL)
CEP: 57072-970 – Maceió – Alagoas – Brasil

{cledsoncalaca, dhs.lima, raphaelpr01}@gmail.com

{eliana.almeida, patrickhenrique}@gmail.com

Abstract. *This paper presents AIIP, an intelligent environment for beginners in programming. AIIP is an environment that aims to help students in their programming learning process. AIIP unites in a single environment for teaching programming desired features by the community, such as a practical teaching module, a theoretical teaching module and an assistent able to act according to user behavior, providing appropriated hints and feedbacks and performs calculations in accordance with pre-established metrics to evaluate the student's solutions. Thus, it is expected that with the combination of these features in one environment, the students could develop skills and reduce their difficults in learning programming.*

Resumo. *Este artigo apresenta o AIIP (Ambiente Inteligente para Iniciantes em Programação). O AIIP é um ambiente que tem o objetivo de auxiliar os alunos em seu processo de aprendizagem de programação. O AIIP une em um só ambiente funcionalidades para o ensino de programação desejáveis pela comunidade da área, tais como: um módulo prático de ensino, um módulo teórico e um assistente capaz de agir de acordo com o comportamento do usuário, fornecendo dicas e feedbacks apropriados, além de realizar cálculos, de acordo com métricas pré-estabelecidas, para avaliar as soluções dos alunos. Desta forma, espera-se que com a junção destas funcionalidades em um único ambiente, o aluno possa desenvolver habilidades e diminuir suas dificuldades de aprendizado de programação.*

1. Introdução

Parece ser consenso entre especialistas da computação que a programação de computadores é uma das disciplinas mais importantes nessa área. Porém, para a maioria dos alunos iniciantes, seu aprendizado e o desenvolvimento de habilidades é um processo complexo, exigente e cercado de dificuldades que podem ser diagnosticadas não somente pelo alto grau de repetência nas disciplinas introdutórias, mas também por deficiências

demonstradas pelos mesmos nas disciplinas mais avançadas e que exigem o pré-requisito de programação [Nobre and Menezes 2002].

Segundo [Gómez-Albarrán 2005] são fatores que afetam negativamente o processo de aprendizagem de programação: a complexidade dos conceitos e estruturas de programação, as dificuldades atribuídas à sintaxe e à semântica das linguagens de programação, bem como a complexidade dos ambientes de programação.

Cientes das dificuldades neste processo, o que realmente podemos observar nos últimos anos é a discussão acerca da utilização de novas metodologias de ensino com objetivo de melhorar o processo de ensino/aprendizagem de programação. Esta discussão tem sido foco de pesquisas no Brasil e no mundo há alguns anos, onde diversas ferramentas e metodologias são propostas tendo como objetivo amenizar tais dificuldades e tornar o processo de ensino e aprendizagem de programação de computadores menos complexo.

Pode-se comprovar o interesse da comunidade no tema no trabalho de [Júnior and Rapkiewicz 2006], no qual verificamos que pelo menos um artigo por ano sobre o ensino de programação é publicado tanto no SBIE como no WEI, eventos ligados à Sociedade Brasileira de Computação. Os autores analisaram vários artigos e concluíram que existem três vertentes na busca por soluções para as dificuldades vivenciadas por professores e alunos neste processo: metodologias, ferramentas computacionais e a integração de ambas. Segundo os autores, apesar de terem encontrado evidências de que a união de ferramentas computacionais e metodologias adequadas culminam em melhores resultados, o que se observa na maioria dos trabalhos é uma tendência em tratá-las separadamente.

Com o intuito de ampliar as discussões sobre o tema, este artigo apresenta um ambiente para o ensino de programação apoiado por um assistente inteligente como proposta para auxiliar tanto alunos como professores com relação ao ensino de programação. Vale ressaltar aqui, que o ambiente não se encontra totalmente implementado, ou seja, alguns componentes citados no decorrer do artigo ainda estão em fase de implementação. Sendo assim, não descrevemos detalhadamente o funcionamento desses componentes por não possuírmos, ainda, conclusões relevantes para discussão, porém, as idéias são expostas para dar ao leitor uma noção de como pretendemos que tais componentes se comportem dentro do ambiente.

O artigo está estruturado da seguinte maneira: a Seção 2 apresenta alguns conceitos que servem como base para o entendimento posterior do artigo e do ambiente apresentado. A Seção 3 apresenta alguns trabalhos relacionados ao ensino e aprendizado de programação. A Seção 4 apresenta o AIIP, incluindo uma visão geral da sua arquitetura, o assistente inteligente e a forma de avaliação dos alunos. Finalmente, a Seção 5 apresenta algumas considerações finais e trabalhos futuros.

2. Embasamento Teórico

Os conceitos apresentados nessa seção dizem respeito à alguns aspectos teóricos relacionados com o tema do artigo, bem como com o desenvolvimento do ambiente a ser apresentado. Através da breve apresentação de tais conceitos será possível um melhor entendimento por parte dos leitores, no que se refere às formas e metodologias de projeto e implementação do ambiente, levando em consideração fatores educacionais relacionados com o ensino de programação.

2.1. Metodologias de Ensino de Programação

Quando falamos sobre o ensino de programação, alguns cuidados básicos por parte dos professores são necessários em diversos fatores, que vão desde o entendimento do problema abordado, do paradigma selecionado, passando pela linguagem escolhida e chegando às metodologias de ensino utilizadas.

Alguns estudos apresentam diferentes soluções metodológicas na tentativa de melhorar o processo de ensino de programação, dentre as quais podemos citar:

Atividades de programação em laboratório: Atividades em laboratório fazem parte de um curso típico de introdução à programação [Robins et al. 2003]. Esta metodologia possui características pedagógicas úteis, uma vez que os alunos podem trabalhar e aprender por conta própria e no seu próprio ritmo, mas não supre todas as necessidades dos alunos, visto que aspectos teóricos da solução do problema muitas vezes não são contemplados, focando apenas a codificação dos algoritmos.

Programação em par: Na programação em par [Han et al. 2010], dois programadores trabalham juntos em um computador no mesmo projeto, algoritmo, código ou teste. É uma metodologia de aprendizagem colaborativa, uma vez que as duas pessoas envolvidas estão trabalhando em conjunto para um objetivo comum [Han et al. 2010]. Esta metodologia é bastante interessante, pois os alunos podem inverter os papeis no processo de aprendizagem, alternando entre codificar e observar a codificação do seu companheiro. Ao mesmo tempo um fator negativo de tal metodologia é que o desempenho do par pode ser afetado de acordo com a discrepância de conhecimento entre os integrantes do par.

Utilização de Tutores de Programação Inteligentes: Tal metodologia também é um meio eficaz e viável para auxiliar programadores iniciantes a superar as dificuldades de aprendizagem. No entanto, sua utilização generalizada tem sido inibida pelos elevados custos e pela complexidade de desenvolvimento associados com a construção de tais sistemas [Pillay 2003].

É importante deixar claro a existência de outras metodologias de ensino de programação, e que, além disso, tais metodologias podem ser combinadas para um uso em conjunto.

2.2. Sistemas Tutores Inteligentes

Técnicas de inteligência artificial aplicadas à ambientes educacionais são geralmente representadas por Sistemas Tutores Inteligentes (`STI`) [Wenger 1987]. Tais sistemas fazem parte de uma classe de programas que possuem certo grau de autonomia de seus módulos e componentes, onde suas funcionalidades variam desde auxiliar o aluno na resolução de um problema até a personalização da assistência provida a um aluno em um determinado momento de sua interação com o ambiente.

No entanto, nem todos os ambientes educacionais são implementados utilizando alguma técnica de inteligência artificial, e mesmo aqueles que se utilizam de alguma técnica de inteligência artificial não necessariamente se enquadram como `STI's` completos, uma vez que a implementação destes sistemas, independente do tamanho do projeto, demandam um grande esforço e um bom tempo, muitas vezes não atingindo os resultados esperados pela comunidade acadêmica da área. Desta forma, a implementação de um

assistente inteligente, menos complexo do que um STI, mas não menos eficaz, pode simplificar o projeto e implementação do ambiente atingindo resultados semelhantes àqueles esperados.

Estamos desenvolvendo e implementando aos poucos para o AIIP, um assistente inteligente capaz de tomar ações de acordo com o comportamento e o estado do usuário no momento de interação com o ambiente. Explicaremos melhor sobre este componente do AIIP na Seção 4.2.

2.3. Arquitetura de Software

A arquitetura de *software*, através de um alto nível de abstração, define o sistema em termos de componentes, a interação entre eles e os atributos e funcionalidades de cada um [Sommerville 2001]. Além disso, a arquitetura representa essa interação de forma explícita, materializando-a através dos conectores. Por conhecerem o fluxo interativo entre os componentes do sistema, essas entidades são capazes, entre outras coisas, de estabelecer protocolos de comunicação e de coordenar a execução dos serviços que envolvam mais de um componente do sistema.

Essa visão estrutural do sistema em um alto nível de abstração proporciona benefícios importantes, que são imprescindíveis para o desenvolvimento de sistemas de *software* complexos. Os principais benefícios são:

i A organização do sistema como uma composição de componentes lógicos;
ii A antecipação da definição das estruturas de controle globais;
iii A definição da forma de comunicação e composição dos elementos do projeto;
iv O auxílio na definição das funcionalidade de cada componente projetado.

Além disso, uma propriedade arquitetural representa uma decisão de projeto relacionada a algum requisito não-funcional do sistema, que quantifica determinados aspectos do seu comportamento, como confiabilidade, reusabilidade e modificabilidade [Sommerville 2001]. Esses conceitos são, de certa forma, melhor entendidos quando na seção 4.1 apresentamos a arquitetura do AIIP e colocamos em prática alguns destes conceitos aqui apresentados, bem como a explicação pela escolha de um determinado padrão de arquitetura e seus benefícios em nosso ambiente.

3. Trabalhos Correlatos

Os ambientes de programação são uma fonte dos obstáculos para os estudantes. Ambientes comerciais são normalmente projetados para serem usados por programadores profissionais. Eles geralmente não fornecem a simplicidade que os estudantes iniciantes precisam, pois normalmente usam apenas um conjunto reduzido das facilidades que eles provêm. [Gómez-Albarrán 2005]

Para enfrentar essas dificuldades, algumas pesquisas têm se concentrado no desenvolvimento de ferramentas com ambientes de desenvolvimento muito simples, que os alunos utilizam facilmente para escrever, compilar, executar e depurar os seus códigos. Essas ferramentas também costumam incluir algum tipo de maior capacidade de diagnóstico. [Gómez-Albarrán 2005]

A análise feita por [Gómez-Albarrán 2005] sintetiza diferentes direções e abordagens para a melhoria do processo de ensino/aprendizagem de programação. O autor

classifica as ferramentas apresentadas em seu trabalho em quatro categorias: (1) Ferramentas que incluem um simples e reduzido ambiente de desenvolvimento; (2) Ambientes baseados em exemplos; (3) Ferramentas baseadas em visualização e animação; e (4) Ambientes de simulação.

Observando os trabalhos citados por [Gómez-Albarrán 2005], e suas consequentes ferramentas e ambientes resultante de suas pesquisas, entendemos que a proposta apresentada neste artigo é bastante interessante por se tratar de uma ferramenta que se enquadra tanto na categoria (2), que incentiva a resolução de problemas de programação, como também por possuir funcionalidades características das categorias (1) e (3), dando mais flexibilidade ao processo de desenvolvimento das habilidades de programação dos alunos.

A proposta apresentada neste artigo se diferencia das muitas apresentadas no trabalho de [Gómez-Albarrán 2005], primeiramente, por usar o português como língua padrão do ambiente, liberando os alunos de qualquer dificuldade na interpretação de mensagens ou avisos. Além disso, utilizamos uma linguagem própria, próxima da linguagem Pascal para o desenvolvimento dos algoritmos, diminuindo a complexidade de aprendizado por parte dos alunos quando comparada ao uso de linguagens como LISP, C e Java, utilizadas em vários trabalhos citados por [Gómez-Albarrán 2005].

Outra contribuição importante da nossa proposta é a flexibilidade tanto na forma de avaliação dos estudantes, quanto nos conteúdos e problemas armazenados no ambiente, que podem sofrer alterações pelo professor a depender de aspectos como: o nível de conhecimento da turma, os conceitos estudados em um dado momento, dentre outros.

4. AIIP: Ambiente Inteligente para Iniciantes em Programação

O AIIP é um ambiente que vêm sendo desenvolvido com o objetivo de auxiliar o processo de ensino/aprendizado de programação. O AIIP tem parte de sua arquitetura (Seção 4.1) baseada no AMBAP [Almeida et al. 2002], uma vez que alguns componentes, como o editor e o interpretador foram adaptados deste ambiente.

Diferentemente do AMBAP, o AIIP possui mais funcionalidades educacionais, por disponibilizar além de um ambiente prático de ensino, a opção de aprendizado teórico onde o aluno pode aprender através de conceitos e exemplos sobre programação, dando também ao aluno a possibilidade de resolver os problemas armazenados em sua base de dados.

O AIIP se utiliza ainda de outros objetos educacionais de aprendizagem no intuito de ajudar o aluno em seu aprendizado, como, por exemplo, a exibição de dicas durante a resolução dos problemas, apresentação de *feedback* a cada problema resolvido, além de um sistema de estatísticas apresentadas ao usuário a cada unidade de conhecimento finalizada, com informações desde o número de exercícios resolvidos dentro da unidade, até uma média final calculada de acordo com parâmetros pré-estabelecidos do sistema, explicados na Seção 4.3.

Além de todas essas funcionalidades, o AIIP possui um assistente inteligente, explicado melhor na Seção 4.2, que acompanha o aluno em toda sua interação com o ambiente, monitorando-o e quando necessário interrompendo-o para informá-lo sobre algum aspecto relevante em seu aprendizado.

4.1. Arquitetura Proposta

A arquitetura proposta para o ambiente está estruturada através de quatro módulos principais, de acordo com o estilo arquitetural em camadas [Shaw and Garlan 1996]. Como citado anteriormente, alguns componentes foram adaptados do AMBAP, como o seu Interpretador e seu Editor, neste caso equivalente ao componente IHC do AIIP. Como pode ser observado detalhadamente na Figura 1, o usuário interage com o sistema através da camada de `Visão`. Essa camada, por sua vez, requisita as funcionalidades da camada de `Sistema`, que é responsável por implementar os casos de uso especificados. Para atender os requisitos especificados, essa camada requisita os serviços especializados da camada de `Negócio`, que contém serviços básicos relativos aos ambientes virtuais de ensino de programação. Finalmente, os dados são armazenados e acessados através da camada de `Dados`.

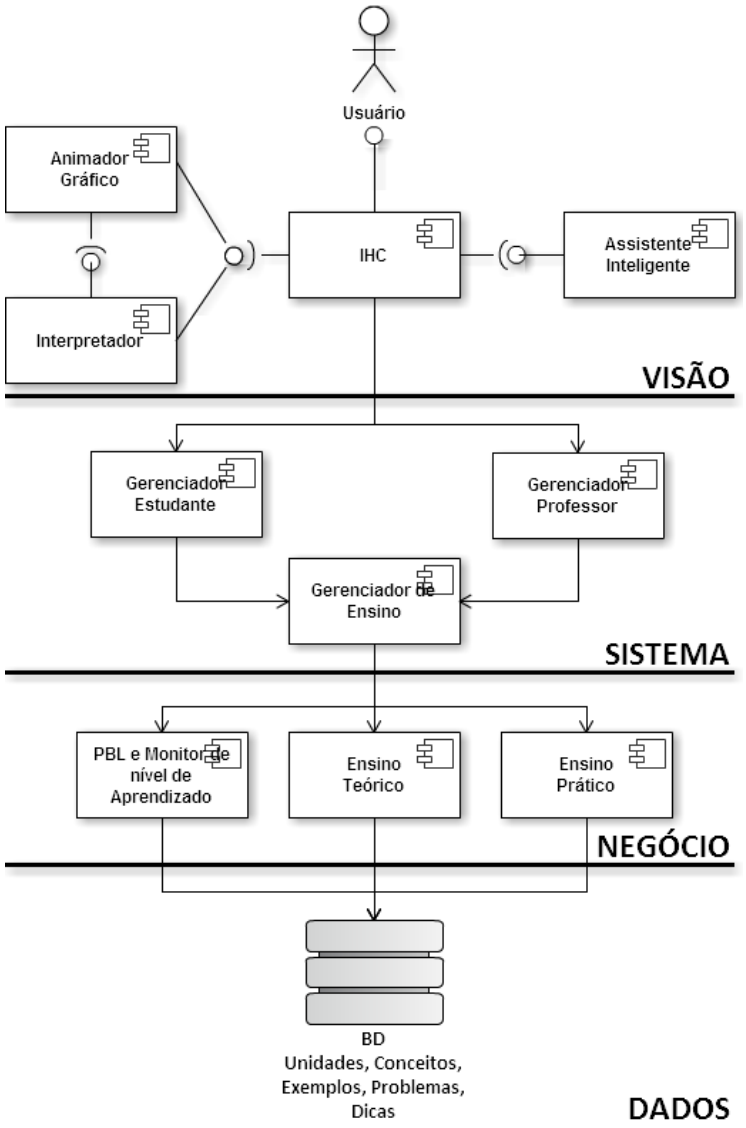

Figura 1. Arquitetura Proposta para o Ambiente

Cada umas dessas camadas está estruturada internamente de acordo com o estilo arquitetural de componentes independentes [Shaw and Garlan 1996], possuindo assim

seus respectivos componentes internos. Esses componentes podem se conectar tanto a outros componentes da mesma camada, quanto a componentes de camadas adjacentes à sua, construindo desta forma a aplicação principal.

Camada de Visão: A camada de `Visão` é responsável por interagir diretamente com os usuários, além de apresentar os diversos tipos de conteúdo que devem ser disponibilizados.

Camada do Sistema: A camada de `Sistema` é responsável pelo gerenciamento das informações dos usuários e dos conteúdos acessados e modificados pelos mesmos. Caso o usuário seja um aluno, ele poderá utilizar o sistema nos modos prático e teórico, tendo suas ações monitoradas pelo sistema. No caso do professor, além dos modos prático e teórico, ele ainda dispõe de um componete para gerenciar todo o conteúdo do sistema e os métodos de avaliação dos níveis de aprendizado dos estudantes.

Camada de Negócio: A camada de `Negócio` tem o objetivo de prover os modos de ensino aos quais o estudante é apresentado na camada de `Visão`. Além disso, esta camada possui um componente responsável por gerenciar a aprendizagem através da resolução de problemas e pelos cálculos e avaliações dos estudantes, de acordo com métricas pré-estabelecidas.

Camada de Dados: A camada mais baixa da arquitetura proposta é a camada de `Dados`, responsável pelo gerenciamento de todo o banco de dados do sistema.

4.2. O Assistente Inteligente

O assistente inteligente do AIIP tem o objetivo de auxiliar os alunos na resolução dos problemas do ambiente, bem como, quando eles estiverem desenvolvendo seus algoritmos, independente de estarem resolvendo um problema específico. Podemos então entender como objetivo principal desse assistente a observação das ações realizadas pelos alunos na resolução de problemas e construção de algoritmos e suas consequentes reações da maneira mais apropriada àquele aluno naquele determinado momento. A partir deste objetivo, suas principais funções são: o fornecimento de dicas e refinamentos das soluções, bem como *feedbacks* apropriados aos alunos, não somente sob demanda, mas também ao perceber a necessidade do aluno, a identificação de erros cometidos pelos alunos bem como os cálculos para avaliação dos algoritmos dos mesmoss.

É importante ressaltar que o assistente do AIIP não se trata de um Sistema Tutor Inteligente - STI [Wenger 1987], visto que não possui todas as características essenciais comuns à estes sistemas, como por exemplo, um módulo de estratégias pedagógicas bem definidas. Porém, a idéia de utilização de um assistente inteligente mais simples ao invés de um STI completo é bastante válida e interessante, uma vez que a construção de STI's em domínios de programação é uma tarefa complexa e que há bastante tempo vem sendo estudada, ainda sem os resultados significativos desejados pela comunidade.

Vale deixar claro aqui que o assistente inteligente do AIIP ainda encontra-se em fase de implementação e testes, ou seja, ainda pode sofrer alterações e melhorias nesta etapa. Consideramos nosso assistente como "inteligente" pelo fato do mesmo ter a capacidade de tomar ações de acordo com regras de produção pré-estabelecidas armazenadas no ambiente, tais como em sistemas especialistas, onde as condições das regras dependem do estado do usuário e as ações são as assistências oferecidas pelo assistente.

Quando o aluno está trabalhando no modo teórico, o assistente não reage às suas ações, porém todo seu progresso com relação aos conteúdos é monitorado pelo ambiente para que posteriormente, na hora de se trabalhar no modo prático, tais informações sirvam como base para que o assistente realize inferências sobre o aprendizado do aluno. Uma vez que o aluno começa a trabalhar no modo prático com objetivo de resolver problemas específicos armazenados no ambiente, o assistente começa realmente a atuar. A ativação do assistente é realizada automaticamente de quatro formas complementares: (1) após cada ação de pressionar a tecla *Enter*, (2) após cada ação de salvar o código, (3) após cada ação de compilar o programa e (4) após uma solicitação específica do aluno.

4.3. Avaliação da Solução dos Problemas dos Alunos

Para avaliar e atribuir uma nota à solução do aluno para os problemas propostos, quatro fatores são analisados pelo assistente inteligente: ***Algoritmo***, ***Dicas***, ***Refinamentos*** e ***Erros***.

Para a atribuição da nota do fator ***Algoritmo***, algumas métricas são levadas em consideração e ponderadas de acordo com sua importância para o código, com relação aos alunos iniciantes em programação, são elas: *Linhas de Código*, *Variáveis*, *Instruções*, *Funções*, *Blocos de Condição* e *Blocos de Repetição*. Cada uma dessas métricas possui um intervalo de aceitação que indica até que ponto a solução do aluno, com relação a cada métrica, está correta. Para um fácil entendimento, no caso da métrica *Linhas de Código* de um problema, onde neste a solução ideal para sua resolução seriam 20 linhas e a solução apresentada pelo aluno possui 30 linhas, a nota desta métrica seria calculada baseada em um índice de aceitação, que neste caso pode ser definida pelo professor de acordo com a especificidade do problema. Ao final de tudo, o assistente realiza um cálculo e atribui uma nota para a solução do aluno de acordo com cada uma dessas métricas, que serão ponderadas com seu peso, para então ser atribuída a nota final do aluno para o fator algoritmo.

Para o fator ***Dicas***, o assistente atribui uma nota levando em consideração os itens: nível de dificuldade do problema, quantidade de dicas da unidade e a quantidade de dicas solicitadas.

Com relação ao fator ***Refinamentos***, que são na verdade os passos necessários para se chegar à resolução de cada problema, o assistente realiza o cálculo da nota de forma semelhante à do fator dicas, sendo que os itens analisados são: nível de dificuldade do problema, quantidade de refinamentos do problema e a quantidade de refinamentos solicitados.

Por fim, o último fator levado em consideração são os ***Erros***. Os erros armazenados são reconhecidos pelo assistente e então, de acordo com o nível de dificuldade do problema e da quantidade de erros cometidos durante aquele problema é realizado um calculo e atribuída uma nota para esse fator.

Ao final da resolução de cada problema é calculada a nota final para o mesmo, de acordo com o peso de cada um dos fatores citados anteriormente, onde o somatório dos pesos dos fatores é igual a 1. Na Tabela 1 é apresentado um exemplo de nota atribuída a um problema solucionado pelo aluno, de acordo com a nota obtida em cada fator e seu peso para o cálculo da nota final.

Tabela 1. Exemplo de nota atribuída ao aluno para um problema

Fator	Nota do Aluno	Peso do Fator	Nota do Fator
Algoritmo	7,00	0,40	2,80
Refinamentos	8,00	0,25	2,00
Dicas	9,00	0,20	1,80
Erros	10,00	0,15	1,50
Nota Final	-	-	**8,10**

É importante ressaltar que, tanto os fatores analisados para compor a nota geral do aluno para um problema, quanto as métricas utilizadas para cada fator, foram elaboradas de acordo com experiências próprias de ensino e da observação e análise de problemas e das dificuldades dos alunos em aulas da disciplina Laboratório de Programação em nossa instituição nos últimos 3 semestres, com aproximadamente 40 alunos por turma/semestre. Também é importante destacar que, de acordo com a preferência do professor, os fatores e as métricas podem ter seus pesos alterados para compor a nota dos alunos, levando em consideração a turma, o problema ou outros aspectos.

Já com a nota para o problema calculada, o aluno então pode trabalhar em outro problema objetivando atingir a pontuação mínima para avançar no conteúdo e estudar conceitos das unidades mais avançadas, e, consequentemente resolver os problemas destas unidades. O aluno deve atingir, no mínimo, 70% da pontuação máxima esperada pela resolução de todos os problemas da unidade para que consiga avançá-la.

Na Tabela 2 esta idéia é exemplificada, onde temos para cada unidade a quantidade de problemas associados a ela, separados por níveis e por pesos, onde o calculo da pontuação máxima de uma unidade é realizado pela soma do resultado da multiplicação entre a quantidade de problemas de cada nível, o peso dos problemas desse nível e 10 (a nota máxima para cada problema). Após a resolução de cada problema, a nota do aluno é armazenada e o assistente verifica se o aluno já tem condições de avançar para unidades posteriores e resolver problemas dessas unidades.

Tabela 2. Exemplo do cálculo da pontuação máxima de cada unidade

Unidade	Qnt Problemas	**Problemas Nível 1**	**Peso Nível 1**	**Problemas Nível 2**	**Peso Nível 2**	**Problemas Nível 3**	**Peso Nível 3**	**Pontuação Máxima Unidade**
1	20	10	3	5	3	5	4	650
2	12	5	3	4	3	3	4	390
3	15	6	3	5	3	4	4	490

5. Considerações Finais e Trabalhos Futuros

Este artigo apresentou o AIIP, um ambiente que vem sendo desenvolvido para auxiliar o ensino de programação, no qual um assistente inteligente interfere no processo de ensino de acordo com o estado e comportamento dos alunos. Algumas métricas foram definidas para que o índice de aprendizagem dos alunos com relação às unidades de conhecimento e aos problemas de programação em que os mesmos são submetidos pudessem ser feitos de forma automática pelo assistente, levando em consideração alguns aspectos de aprendizagem.

O ambiente pode auxiliar não só os alunos, mas também aos professores, uma vez que ao acompanhar a evolução, oferecer feedbacks e dicas e avaliar as soluções dos alunos automaticamente no momento da interação com os mesmos, o AIIP libera, parcialmente,

o professor para dedicar mais atenção e um acompanhamento maior para alunos com mais dificuldades na turma. Outro ponto importante é a flexibilidade que o professor possui para alterar os índices das métricas de avaliação dos alunos, dependendo do nível de conhecimento da turma ou da dificuldade de alguns problemas.

Desta forma, acreditamos que do modo que nosso ambiente foi estruturado, a junção de objetos de ensino Teórico e Prático, apoiados por um assistente inteligente, estimulando a resolução de problemas, formam um conjunto de funcionalidades lúdicas que atendem, até certo ponto, aos requisitos esperados pela comunidade da área no processo de ensino/aprendizagem de programação.

Pretendemos, tão logo quanto possível e após a implementação total de todos os componentes, publicar os resultados de avaliações realizadas junto aos alunos ingressantes no curso de Ciência da Computação da nossa instituição com respeito à satisfação e níveis de dificuldade e aprendizagem dos conceitos de programação com a utilização do ambiente, bem como relatar as melhorias dos componentes que ainda estão em fase de implementação para que o mesmo possa ser avaliado de forma mais completa.

Referências

Almeida, E. S., Costa, E. B., Silva, K. S., Paes, R. B., Almeida, A. A. M., and 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.

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

Han, K., Lee, E., and 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.

Júnior, J. C. R. P. and 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.

Nobre, I. A. M. and 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.

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

Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: a review and discussion. *Computer Science Education*, 13:137–172.

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

Sommerville, I. (2001). *Software Engineering*. Addison-Wesley, 6th edition.

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.