Analogus: Um Ambiente para Auxílio ao Ensino de Programação Orientado pelo Raciocínio por Analogia

Gilson P. dos Santos Júnior[1], **Joseana Macêdo Fechine**[1], **Evandro de Barros Costa**[1,2]

[1]Programa de Pós-Graduação em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande (UFCG)
Caixa Postal 10.106 – 58.109-970 – Campina Grande – PB – Brasil

[2]Instituto de Computação – Universidade Federal de Alagoas (UFAL)
Maceió – AL – Brasil

{gilson,joseana}@dsc.ufcg.edu.br, ebcosta@gmail.com

Abstract. *Learning to solve programs using algorithms is one of the main challenges of the introductory programming courses, once beginners find difficult to use one of the most used way of thinking by expert programmers: analogy-based reasoning. In order to help beginner programmers to start using this solving approach, we have developed Analogus. It is a programming environment which helps students to identify previously solved problems which are similar to the current one, using a RBC engine alongside with a chatterbox that helps them to think about similarities aspects.*

Resumo. *Aprender a solucionar problemas algoritmicamente é um dos principais desafios no aprendizado das disciplinas de introdução à programação, uma vez que os alunos iniciantes sentem dificuldades em empregar uma das principais formas de raciocínio utilizadas pelos programadores experientes: raciocínio por meio de analogia. Para auxiliar o aluno no aprimoramento das habilidades de resolução de problemas por meio de analogias foi desenvolvido o Analogus. O Analogus é um ambiente de resolução de problemas de programação que ajuda ao aluno a identificar problemas resolvidos similares ao atual, por meio de um raciocinador baseado em casos, ao mesmo tempo um chatterbot o auxilia a refletir sobre os aspectos de similaridade.*

1. Introdução

O aprendizado de programação é uma tarefa complexa que apresenta inúmeros desafios para os alunos iniciantes, uma vez que os alunos devem aprender a solucionar problemas algoritmicamente, aprender uma nova linguagem (sintaxe e semântica), utilizar um ambiente de programação e testar e depurar programas.

Pesquisadores apontam que solucionar problemas por meio de algoritmos é um dos desafios que apresenta maior grau de dificuldade para os alunos iniciantes [Delgado 2005, Muller 2005, Haberman and Muller 2008], já que eles sentem dificuldades em analisar problemas e formular soluções.

Tal situação é agravada pela abordagem de ensino empregada nas disciplinas de introdução à programação que, na maioria das universidades, possuem ementas cur-

riculares que focam unicamente no aprendizado da sintaxe e semântica de linguagens [Muller 2005].

Contudo, no domínio de programação, uma das principais abordagens de resolução de problemas é o raciocínio por analogia, comumente observado no cotidiano dos profissionais de informática, visto que programadores experientes recorrem a soluções passadas e as adaptam para solucionar novos problemas, evitando, desta forma, a criação de uma solução a partir do zero [Bergin et al. 2001, Muller 2005, Muller et al. 2007, Oliveira et al. 2008].

Embora o raciocínio por analogia seja natural para os programadores experientes, alunos iniciantes sentem dificuldade em identificar semelhanças entre os problemas apresentados nas disciplinas de introdução à programação [Muller 2005, de Barros et al. 2005] e, consequentemente, em empregar o raciocínio por analogia para solucioná-los.

É notável que durante o curso, alguns alunos conseguem aprimorar sozinhos a habilidade de resolução de problemas por analogia e alcançar o nível de programadores experientes. Mas, o caminho enfrentado por eles é constituído por dificuldades que, muitas vezes, leva a reprovação nas disciplinas de programação e/ou, até mesmo, à desistência do curso.

É inegável que a utilização de ferramentas e metodologias de ensino de programação adequadas pode mitigar o esforço dos alunos iniciantes no desenvolvimento dessa habilidade. Devido a isso, ao longo dos anos, muitas ferramentas têm sido desenvolvidas para o ensino de programação, porém ainda existe uma carência de ferramentas que auxiliem os alunos na identificação de similaridades entre os problemas e deem suporte à resolução de problemas por analogia.

Esse artigo apresenta o ambiente de resolução de problemas de programação *Analogus*, cujo objetivo é auxiliar o aluno iniciante no aprimoramento da habilidade de resolução de problemas por analogia. A ferramenta auxilia o aluno na identificação de similaridades entre o problema atual e os previamente solucionados, e dá suporte a resolução do problema atual.

As demais seções deste artigo estão estruturadas da seguinte forma: na Seção 2 são apresentados os trabalhos relacionados; na Seção 3 é exposta a visão geral do ambiente *Analogus* e; na Seção 4 são apresentadas as considerações finais e os trabalhos futuros.

2. Trabalhos Relacionados

Ao longo dos anos, inúmeras ferramentas e metodologias de ensino foram desenvolvidas buscando maximizar o aprendizado de programação nas disciplinas introdutórias. Porém, poucos são os trabalhos que focam o aprimoramento da habilidade do aluno iniciante de solucionar problemas, visando tornar o aluno não apenas um bom programador, mas um bom resolvedor de problemas.

No contexto de metodologias de ensino de programação que visam o aperfeiçoamento dessa habilidade nos alunos é possível citar a baseada na instrução de padrões de programação para alunos iniciantes [Proulx 2000]. Tais metodologias pregam que os alunos devem aprender a programar por meio de padrões e, a partir da instanciação e combinações deles, solucionar os problemas de programação.

Os padrões de programação são abstrações de trechos de código que descrevem soluções para problemas análogos, além de bons exemplos de soluções elegantes e eficientes, desenvolvidas, geralmente, por especialistas [Muller et al. 2004].

Seguindo a linha das metodologias de ensino de padrões para iniciante, é importante citar a abordagem *Pattern-Oriented Instruction* (POI). O POI é uma abordagem pedagógica que incorpora padrões algorítmicos no ensino introdutório de programação. O grupo de ensino de programação da Universidade de Tel-Aviv publicou inúmeros trabalhos [Muller et al. 2004, Muller 2005, Muller et al. 2007, Muller 2008, Haberman and Muller 2008] que fundamentam, justificam e comprovam a eficiência da utilização dessa abordagem metodológica em sala de aula para o desenvolvimento da habilidade do aluno de solucionar problemas de programação por meio do raciocínio por analogia. Embora o grupo possua importantes publicações nessa linha, seu foco é estritamente pedagógico e, até o presente momento, nenhuma ferramenta computacional que auxilie a aplicação dessa metodologia foi apresentada. Em contra partida, é possível encontrar algumas ferramentas que auxiliam o ensino de programação por meio dos padrões elementares como é o caso do ProPAT [de Barros et al. 2005] e o SELP [Oliveira et al. 2008].

Ainda que as ferramentas citadas possuam a característica de ajudar na instrução das disciplinas de introdução focada em padrões, visando ensinar o aluno a solucionar problemas de programação por analogia, em nenhuma delas há mecanismos que assessore o aluno iniciante na identificação de problemas similares ao que está resolvendo, como ocorre no ambiente de ensino de lógica de programação, desenvolvido por Koslosky [Koslosky 1999], e na ferramenta de construção de abstrações em lógica de programação de Mattos [Mattos 2002].

Os trabalhos de Koslosky e Mattos fornecem ambientes de ensino/aprendizagem de programação, assistidos por um sistema de raciocínio baseado em casos (RBC), que auxilia o aluno a construir soluções para o problema atual a partir das soluções passadas, recuperadas pelo RBC.

Embora essas ferramentas recuperarem soluções de problemas de programação similares ao atual, as informações utilizadas em sua recuperação são pouco precisas. Em ambos os trabalhos, o algoritmo calcula a similaridade entre os problemas a partir do enunciado, da quantidade de instruções e variáveis e da quantidade de estruturas de repetição e seleção. Além disso, em nenhuma das ferramentas analisadas é observado um mecanismo de interação com o aluno que o ajude a perceber os aspectos que tornam os problemas recuperados semelhantes ao atual, nem os mecanismos de auxílio ao ensino por meio de padrões de programação.

3. O *Analogus*

O *Analogus* é um ambiente de resolução de problemas de programação que visa aprimorar a habilidade do aluno iniciante em solucionar problemas por meio do raciocínio por analogia. Para isso, essa ferramenta apresenta duas importantes características:

- Recuperar os problemas previamente resolvidos pelo aluno, similares ao atual;
- Ajudar o aluno iniciante a refletir sobre os aspectos que tornam tais problemas semelhantes.

Assim que um novo problema de programação é sugerido ao aluno um sistema de raciocínio baseado em casos recupera automaticamente de uma base um conjunto de problemas previamente resolvidos pelo aluno, similares ao atual. No instante em que os problemas são recuperados, um *chatterbot*[1] inicia um diálogo com o aluno iniciante, discutindo os aspectos que levam tais problemas a serem semelhantes ao atual.

Diante dos problemas recuperados e do diálogo com o *chatterbot*, o aluno cria a solução do novo problema e a submete para a avaliação do professor. O professor, por sua vez, efetua comentários sobre a solução. Assim que todas as considerações do professor são atendidas, a solução é armazenada na base de casos do *Analogus*. Na Figura 1 está ilustrado o processo descrito.

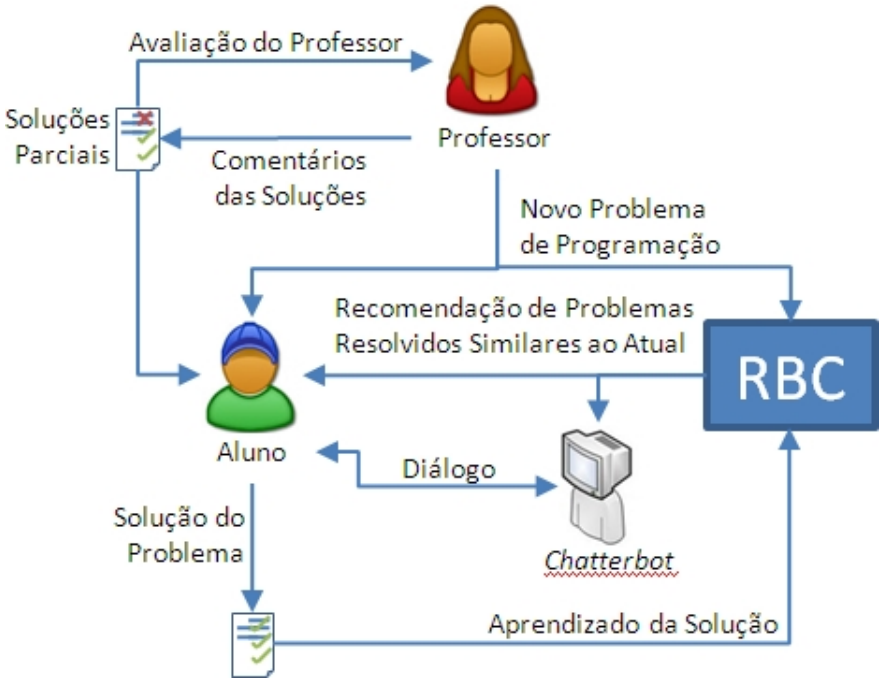

Figura 1. Visão Geral do *Analogus*.

Para atender essas características, o *Analogus* possui uma arquitetura cliente-servidor, conforme ilustrado na Figura 2. O cliente é uma interface *Web*, implementada utilizando a tecnologia *Google Web Toolkit* (GWT). Já o servidor é formado por três módulos: o módulo RBC, o módulo *chatterbot* e o módulo interpretador de *Python*. O módulo RBC é responsável pela implementação do sistema de raciocínio baseado em casos para o domínio de problemas de programação. O módulo *chatterbot* é responsável pela implementação do sistema de diálogo que ajuda o aluno a refletir sobre os problemas similares. Ao passo que, o módulo do interpretador tem a função de se comunicar com um interpretador da linguagem *Python* e retornar o resultado da execução do programa.

O *Analogus* é indicado para alunos iniciantes na resolução das atividades práticas das disciplinas de introdução a programação, principalmente, nos cursos que visam tornar os alunos bons resolvedores de problemas de programação. A linguagem de programação escolhida para o ensino de programação no *Analogus* é *Python*.

[1] Os *chatterbots* são programas capazes de simular uma conversação com um ser humano, com o objetivo de fazer o interlocutor pensar que está falando com outro humano [Leonhardt et al. 2007].

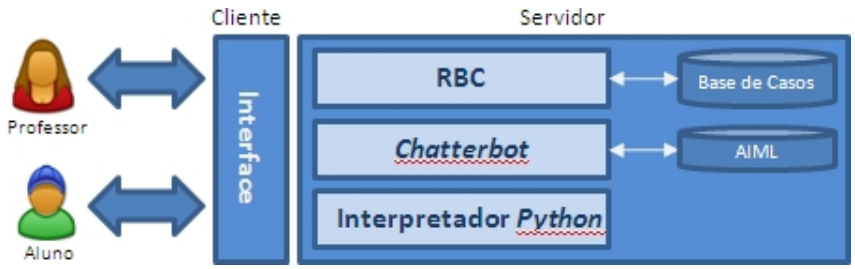

Figura 2. Arquitetura do *Analogus*.

É importante ressaltar que essa é uma ferramenta de auxílio ao ensino/aprendizagem de programação, logo, é fundamental seu emprego aliado a uma metodologia de ensino de padrões para alunos iniciantes.

A seguir, nas Seções 3.1 e 3.2, serão descritos os módulos RBC e *Chatterbot*, apresentados na arquitetura (Figura 2) do *Analogus*. As telas e suas funcionalidades serão apresentadas na Seção 3.3.

3.1. Módulo RBC

O módulo RBC tem a função de instanciar um sistema de raciocínio baseado em casos para o domínio de problemas de programação. Para a implementação desse módulo foi utilizado o *framework* JColibri 2.0 [Agudo et al. 2007, Recio-García et al. 2008]. Esse *framework* é *open-source*, orientado a objetos e foi desenvolvido em Java pelo grupo GAIA[2] para criação de raciocinadores baseados em casos.

O raciocínio baseado em casos é uma técnica da Inteligência Artificial inspirada no raciocínio por analogia, que tenta resolver novos problemas adaptando soluções passadas. O ciclo clássico de um sistema de raciocínio baseado em casos foi proposto por [Aamodt and Plaza 1994]. Este ciclo é composto por 4 (quatro) fases, sendo estas:

- Recuperação - recupera um conjunto de casos similares da memória;
- Reuso - adapta os casos recuperados a fim de solucionar o problema atual;
- Revisão - revisa e testa a solução proposta;
- Retenção - armazena a solução revisada;

Um caso, por sua vez, é um registro de uma experiência, uma determinada situação. Nesse domínio, o caso foi representado pelo seu enunciado, pelos padrões de programação sugeridos pelo professor para solucioná-lo, pela complexidade e pela categoria. Nesse trabalho, os problemas de programação são classificados em problemas matemáticos, de jogos ou de sistemas de informação, conforme [Barreto et al. 2008]. A solução do caso é o algorítmo desenvolvido pelo aluno para solucionar tal problema. O diagrama de classes apresentado na Figura 3 está ilustrada a representação do caso.

Os padrões para alunos iniciantes de programação foram adicionados na representação do caso, uma vez que a solução de um problema de programação pode ser construída a partir da combinação de padrões, aninhando ou encadeando, e alguns trabalhos apontam os padrões como um importante elemento na utilização do raciocínio por analogia para resolução de problemas de programação [Muller et al. 2004, Muller 2005].

[2]GAIA (*Group for Artificial Intelligence Applications*) Grupo do Departamento de Engenharia de Software e Inteligência Artificial da Universidade de Madrid.

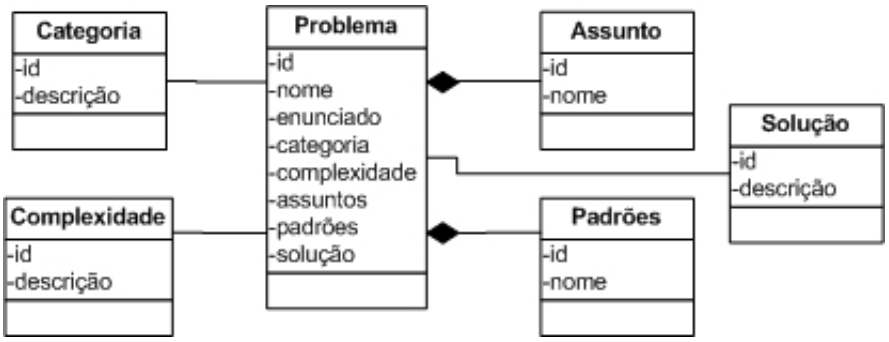

Figura 3. Representação do caso

Na implementação da fase de recuperação foi utilizado o algorítmo *k-Nearest Neighbors* (KNN). O KNN é um algorítmo que calcula a similaridade entre dois casos a partir da média ponderada das similaridades locais, ajustadas pelos pesos de cada atributo. As funções de similaridade local calcula a similaridade entre dois atributos e dependem da sua natureza. Dessa forma, foi utilizada a função *equal()*, na qual dois atributos são similares se possuem o mesmo valor, e a função *cosineCoefficient()*, que calcula o cosseno do ângulo entre dois vetores de n dimensões. Nessa função, dois atributos são similares se o valor do cosseno é igual a um.

A fim de efetuar o ajuste ideal dos pesos para cada atributo representado no caso, foi realizado um experimento, no qual foram testadas combinações de pesos conforme descrito em [Santos et al. 2008].

No reuso, o aluno se baseia nos problemas recuperados e implementa um solução inicial para o problema. Essa solução é enviada para avaliação do professor que efetua os devidos comentários à solução do aluno (fase de revisão). Assim que todos os comentários do professor foram atendidos pelo aluno e a solução esteja finalizada, o ambiente armazena na base de casos esse novo caso. Dessa forma, o ciclo do RBC é finalizado.

3.2. Módulo *Chatterbot*

Esse módulo tem como objetivo desenvolver um *chatterbot* para dialogar com o aluno sobre as similaridades entre os problemas recuperados pelo RBC.

A arquitetura de um *chatterbot* é constituída por uma *interface* gráfica, que interage diretamente com o interlocutor; uma máquina de inferência, que analisa a pergunta realizada pelo interlocutor e infere uma resposta a partir de uma base de conhecimento; e a base de conhecimento que é o cérebro do *chatterbot*.

O AIML é uma linguagem de marcação utilizada para definição da base de conhecimento de *chatterbots* [Pirrone et al. 2008]. Essa linguagem permite a definição de estímulos, *patterns*, e a correspondente resposta do sistema, *template*.

Durante anos, vários *chatterbots* foram criados com intuíto de auxiliar o ensino/aprendizagem do aluno como, por exemplo, a Doroty [Leonhardt et al. 2007], utilizada no treinamento de gerentes de redes pouco experientes, ou a Elektra [Leonhardt et al. 2003], uma professora de Física virtual. O emprego dos *chatterbots* na educação à distância talvez seja uma das áreas mais promissoras, uma vez que esse agente está disponível a todo instante para tirar dúvidas dos alunos.

Na implementação desse módulo foi utilizado o *framework Chatterbean*. O *Chatterbean* é um interpretador AIML *open-source*, orientado a objetos e foi desenvolvido em Java.

O *chatterbot* do ambiente *Analogus* apresenta três características marcantes são elas:

- Discutir com o aluno sobre os aspectos de similaridade entre os problemas resolvidos;
- Retirar dúvidas sobre a linguagem de programação (Python) e os padrões ensinados em sala de aula pelo professor.
- Interface animada, representando o estado emocional do *chatterbot*. No *Analogus*, o *chatterbot* pode apresentar as expressões faciais de tristeza, de tranquilidade, de felicitação e de alegria.

Para discutir com o aluno sobre os problemas similares e suas características, o Módulo do *Chatterbot* se comunica com o Módulo RBC e obtêm informações como, por exemplo: Quais problemas são similares ao atual? Qual o nível de similaridade de cada atributo em relação ao problema atual (Categoria, Enunciado e Padrões de Programação)? A seguir um trecho de diálogo entre o *chatterbot* e o aluno é descrito.

- Professor: Olá minha cara aluna! Vamos resolver o Problema A? (SIM/NÃO)
- Aluno: Sim
- Professor: Ótimo! Nesse problema temos que 'Escreva um programa que imprime uma tabela em que cada linha contém: um inteiro e sua potência de 2. O número de linhas a produzir deve ser pedido ao usuário. Os números da primeira coluna devem estar justificados à esquerda e as potências à direita. Entre as duas colunas deve haver apenas um espaço em branco.'. Você se recorda de ter resolvido algum problema semelhante ao Problema A? (SIM/NÃO)
- Aluno: Sim
- Professor: Então, por favor, selecione na janela de 'Problemas Resolvidos', um problema que você julga semelhante ao Problema A.
- Professor: Hum... Você selecionou o problema: 3. Vamos analisar as semelhanças entre o problema atual e o selecionado. Quais características você julga semelhante entre o Problema A e o 3? (ENUNCIADO, CATEGORIA DO PROBLEMA, PADRÕES DE PROGRAMAÇÃO).
- Aluno: Enunciado do Problema
- Professor: Vejamos os enunciados dos problemas... Interessante!!! Pelo enunciado eu considero que estes problemas são 'bem semelhantes'.
- Professor: Alguma outra característica você julga semelhante entre o Problema A e o Problema 3? (CATEGORIA DO PROBLEMA, PADRÕES DE PROGRAMAÇÃO).
- Aluno: Os padrões de programação.
- Professor: Quais padrões você acredita serem utilizados na solução de ambos os problemas?
- Aluno: Padrão de repetição contada.
- Professor: Parabéns!!! Ambos os problemas utilizam o padrão de repetição contada na sua solução.

As expressões faciais do *chatterbot* variam no decorrer do diálogo. Quando o *bot* questiona sobre a similaridade entre dois problemas e o aluno responde corretamente, o *chatterbot* muda sua expressão facial para felicitação e, em seguida, retorna a feição de tranquilidade. Por outro lado, quando o aluno comete sucessivos erros, o *bot* apresenta a expressão facial de tristeza.

3.3. Interfaces e Funcionalidades do *Analogus*

O *Analogus* apresenta duas visões distintas, a visão do professor e a visão do aluno.

- Visão do Professor - fornece as interfaces de cadastro de problemas, cadastro de padrões, revisão das soluções submetidas pelo aluno;
- Visão do Aluno - permite ao aluno escolher qual problema será resolvido; visualizar o enunciado do problema, as soluções dos problemas resolvidos, os comentários do professor sobre a solução do problema atual e os padrões de programação ensinados em sala de aula; codificar a solução; executar o problema; e discutir os aspectos de similaridade com o *chatterbot*.

Na Figura 3.3 está ilustrada a Visão do Aluno, no momento em que é questionado sobre quais problemas são similares ao atual.

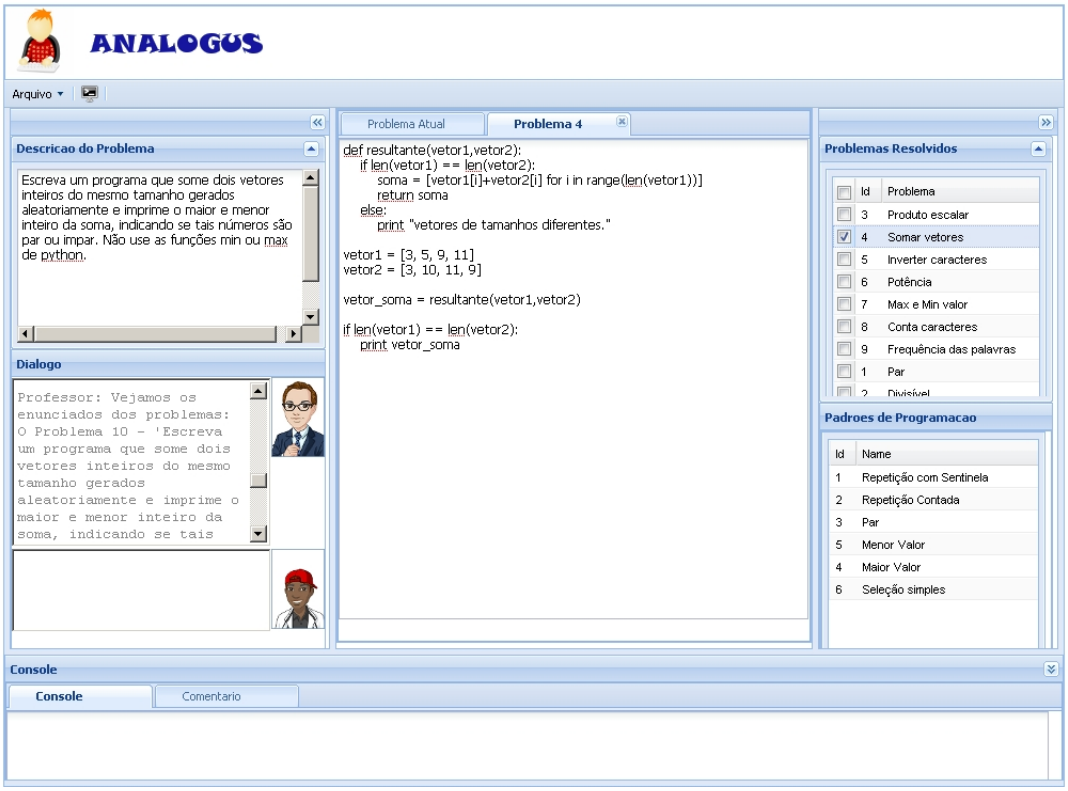

Figura 4. Visão do Estudante no *Analogus*

4. Conclusão

O desenvolvimento de ferramentas e metodologias de ensino para maximizar o aprendizado de programação tem sido, durante anos, um campo da informática na educação

amplamente pesquisado. Mesmo assim, ainda hoje, poucas são as ferramentas computacionais com objetivo de aprimorar a habilidade do aluno em solucionar problemas por meio do raciocínio por analogia.

Nesse sentindo, algumas ferramentas como o ProPAT, o SELP, o ambiente de programação proposto por Koslosky e o sistema de Mattos merecem destaque, embora nenhum deles seja capaz de indicar problemas similares ao mesmo tempo em que favorece a reflexão do aluno sobre as similaridades entre os problemas.

O *Analogus* tem como maior contribuição minimizar as dificuldades apresentadas pelo aluno na identificação dos problemas similares ao que está resolvendo, ao mesmo tempo em que favorece a reflexão sobre os aspectos que tornam tais problemas semelhantes, por meio de um agente inteligente de diálogo.

A linguagem Python foi escolhida para ser utilizada nessa versão do ambiente, devido ao crescimento de interesse na comunidade por essa linguagem para o ensino de programação para alunos iniciantes e por permitir uma posterior avaliação na Universidade Federal de Campina Grande - PB, onde está sendo planejado um conjunto de experimentos com uma turma iniciante.

Futuramente, é interessante ampliar a base de problemas do *Analogus* e possibilitar que os alunos colaborem sua base de casos entre si, trocando experiências e, consequentemente, favorecendo a aprendizagem colaborativa.

Referências

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approach. 7:1:39–59.

Agudo, B. D., Pedro, Juan, and Sánchez, A. (2007). Building cbr systems with jcolibri. *Special Issue on Experimental Software and Toolkits of the Journal Science of Computer Programming*, 69(1-3):68–75.

Barreto, E., Romao, M., Neto, F., Mendonça, A., Santos, G. P. S., and Fechine, J. M. (2008). Uma aplicação de algoritmos genéticos interativos para composição de listas de problemas de programação.

Bergin, J., Brady, A., Duvall, R., Proulx, V., and Rasala, R. (2001). Using patterns in the classroom. In *CCSC '01: Proceedings of the sixth annual CCSC northeastern conference on The journal of computing in small colleges*, páginas 5–7, , USA. Consortium for Computing Sciences in Colleges.

de Barros, L. N., Ana, Delgado, K. V., and Matsumoto, P. M. (2005). A tool for programming learning with pedagogical patterns. In *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, páginas 125–129, New York, NY, USA. ACM.

Delgado, K. V. (2005). Diagnóstico baseado em modelos num sistema tutor inteligente para programação com padrões pedagógicos. Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo.

Haberman, B. and Muller, O. (2008). Teaching abstraction to novices: Pattern-based and adt-based problem-solving processes. In *38th ASEE/IEEE Frontiers in Education Conference*.

Koslosky, M. A. (1999). Aprendizagem baseada em casos - um ambiente de ensino de programação. Master's thesis, Universidade Federal de Santa Catarina.

Leonhardt, M. D., Castro, D. D., Dutra, R. L., and Tarouco, L. M. (2003). Elektra: Um chatterbot para uso em ambiente educacional. 1.

Leonhardt, M. D., Tarouco, L., Vicari, R. M., Santos, E. R., and Dos (2007). Using chatbots for network management training through problem-based oriented education. In *Advanced Learning Technologies, 2007. ICALT 2007. Seventh IEEE International Conference on*, páginas 845–847.

Mattos, M. M. (2002). Learning how to build abstractions in programming logics classes. 6.

Muller, O. (2005). Pattern oriented instruction and the enhancement of analogical reasoning. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, páginas 57–67, New York, NY, USA. ACM.

Muller, O. (2008). Developing algorithmic problem-solvin skills - the rationale underlying a course.

Muller, O., Ginat, D., and Haberman, B. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, páginas 151–155, New York, NY, USA. ACM.

Muller, O., Haberman, B., and Averbuch, H. (2004). (an almost) pedagogical pattern for pattern-based problem-solving instruction. In *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, páginas 102–106, New York, NY, USA. ACM.

Oliveira, C. H. J. S., Greghi, R. R., and Pimentel, E. P. (2008). Ambiente para assistência à aprendizagem e programação baseado em padrões pedagógicos.

Pirrone, R., Cannella, V., and Russo, G. (2008). Gaiml: A new language for verbal and graphical interaction in chatbots. In *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on*, páginas 715–720.

Proulx, V. K. (2000). Programming patterns and design patterns in the introductory computer science course. In *SIGCSE '00: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, páginas 80–84, New York, NY, USA. ACM.

Recio-García, J. A., Díaz-Agudo, B., and González-Calero, P. (2008). Jcolibri2 tutorial. Technical report.

Santos, Costa, E., and Fechine, J. M. (2008). Raciocínio baseado em casos para auxílio a alunos na resolução de problemas por analogia - uma abordagem para representação e recuperação de casos.