Conduzindo os alunos a resolverem exercícios de programação através da verificação de seus objetivos durante a execução do programa

Adilson Vahldick[1], Gabriel Naoto Ymai Pereira[1]

[1]Departamento de Engenharia de Software, Centro de Educação Superior do Alto Vale do Itajaí, Universidade do Estado de Santa Catarina (Udesc) – Ibirama, SC – Brasil

adilson.vahldick@udesc.br, naoto.ymai@gmail.com

Abstract. *Students in an Introductory Programming class of a Software Engineering course had used a ludic framework to learn programming in place of traditional methodology. On the other hand, the exercises were more sophisticated, practically demanding in all exercises that the student had to first plan the solution and then solve with programming. During the programming exercises, students with difficulties in resolving them need to have the initiative to ask the teacher for help. Even the most experienced may end up giving up solving the exercise because of some difficulty. To keep them motivated to continue, this work presents the inclusion of a new feature in the ludic framework that lists all the objectives of the exercise and, while executing the student's solution, verification is made in the fulfilment of these objectives. This feature can lead the student to solve the problem in parts, and thus remain motivated to complete the exercise. This work presents the ludic framework and demonstrates the goal verifier. An experiment with 16 students was conducted, and the results showed that students who used this verifier resolved more exercises than those who did not use the feature.*

Resumo. *Os alunos da disciplina de Introdução à Programação de um curso de Engenharia de Software vinham utilizando um framework lúdico para aprender programação em substituição à metodologia tradicional. Em contrapartida, os exercícios eram mais sofisticados, praticamente exigindo em todos os exercícios que o aluno tivesse que realmente planejar a solução para depois resolver com programação. Durante os exercícios de programação, os alunos com dificuldades em resolvê-los precisam ter iniciativa para solicitar auxílio ao professor. Mesmo os mais experientes podem acabar por desistir de resolver o exercício por conta de alguma dificuldade. Para mantê-los motivados em prosseguir, esse trabalho apresenta a inclusão de um novo recurso no framework lúdico que lista todos os objetivos do exercício e, enquanto executa a solução do aluno, é feita a verificação no cumprimento desses objetivos. Esse recurso pode conduzir o aluno a resolver o problema em partes menores, e assim manter-se motivado em concluir o exercício. Esse trabalho apresenta o framework lúdico e demonstra o verificador de objetivos. Foi realizado um experimento com 16 alunos, e os resultados demonstraram que aqueles que usaram esse verificador resolveram mais exercícios do que os alunos que não usaram o recurso.*

DOI: 10.5753/cbie.sbie.2019.71

1. Introdução

Aprender programação de computadores não é uma tarefa tão simples para muitos estudantes. A motivação, persistência, confiança, responsabilidade emocional, estratégias de resolução de problemas são alguns dos fatores que podem influenciar a apropriação de competências necessárias (WINSLOW, 1996). A complexidade desse processo faz dele fonte de dificuldades na sua aprendizagem, e por consequência na alta desistência e reprovação nessas disciplinas nos primeiros anos dos cursos superiores (ROBINS; ROUNTREE; ROUNTREE, 2003).

Os estudantes da nova geração, habituados com jogos e outras mídias eletrônicas, não se sentem motivados com exercícios para calcular e imprimir números no console ou em uma janela, mas estão acostumados a consumir animações, gráficos e sons com ritmo acelerado, e provavelmente esse é um tipo de mídia que eles gostariam de produzir (GUZDIAL; SOLOWAY, 2002). Em Vahldick e Mattos (2008) é relatado o desenvolvimento e uso de um *framework* lúdico chamado Furbot para oferecer exercícios de programação em Java para alunos de Ciência da Computação. Entretanto, o *framework* acabou por ser migrado para o ensino de Pensamento Computacional para crianças de 8 anos (ARAÚJO; SILVEIRA; MATTOS, 2018). Contudo, com o código fonte do projeto original disponível em um repositório público, foi possível evoluir para um novo projeto, atualizando sua interface gráfica, resolvidos alguns erros e melhoradas suas classes acessíveis aos alunos. O novo *framework* foi rebatizado de AgenteJ.

Para que o aprendizado seja efetivo, o aluno precisa estar motivado, através de um ambiente amigável, o conteúdo interessante e pela didática do professor (BERGIN et al., 2001). Ainda, segundo esses mesmos autores, para construir e manter a confiança do aluno, é preciso prepará-los para que desempenhem as tarefas de forma autônoma, apresentando objetivos claros, que possam conduzi-los no cumprimento dessas tarefas. Escrever programas envolve compreender o problema e abstrair um modelo, dividir o problema em partes menores, decidir pela melhor estratégia para resolver cada uma das partes, e aplicar ou adaptar soluções já conhecidas (ROBINS; ROUNTREE; ROUNTREE, 2003). No intuito de conduzir as tarefas dos alunos com o *framework* AgenteJ, foi adicionado um recurso para que apresente os objetivos parciais a serem cumpridos em cada um dos exercícios, e quando a classe for executada, esses objetivos são verificados e validados, fazendo com que o aluno tenha noção da sua direção tomada na resolução do exercício.

Algumas pesquisas sugerem um processo automatizado de verificação de código e avaliação das soluções por parte do professor (BARBOSA et al., 2016; SANTOS; SEGUNDO; TELVINA, 2017). Entretanto, além de analisar o código, é preciso conduzir o aluno durante a própria tarefa para alcançar a solução. Existem trabalhos que propõem o suporte ao aluno com a mediação do professor (KUTZKE; DIRENE, 2014; LOPES; GOMES; DANTAS, 2017). Porém, o trabalho aqui apresentado propõe que durante a própria resolução do exercício, o aluno tenha uma maneira de saber o que ele já cumpriu e o que falta para encerrar o problema.

Esse artigo apresenta esse novo recurso no *framework* e sua avaliação com 16 alunos da disciplina de Introdução à Programação do curso de Bacharelado de Engenharia de Software da Universidade do Estado de Santa Catarina. O trabalho está estruturado da seguinte maneira: na seção 2 é apresentado o *framework* Furbot e sua

nova versão AgenteJ destacando o recurso de apresentação e validação dos objetivos parciais. A seção 3 detalha o experimento e a quarta apresenta os resultados. As conclusões são apresentadas na quinta seção.

2. Framework revisado: AgenteJ

O AgenteJ é uma nova versão do Furbot. O Furbot caracterizava-se por um *framework* concebido para tentar diminuir as dificuldades de aprendizagem e ensino na lógica de programação através de um forte apelo à área de jogos, criando assim uma atmosfera facilitadora ao aprendizado (ESTRÁZULAS et al., 2009). Vahldick e Mattos (2008) afirmam que quando o aluno programa o robô do Furbot para explorar o mundo, contar e guardar os objetos que ele vai encontrando, é como se ele estivesse programando um jogo. A Figura 1 exemplifica como um robô pode ser implementado utilizando o *framework*, mostrando também que ele permite que o aluno utilize métodos implementados na classe Furbot, como por exemplo, obter informações sobre o ambiente (*ehVazio(Direcao)*) e movimentar (*andarAcima()*) o robô. As definições do exercício estão no arquivo Mundo02.xml. Nesse arquivo se define a dimensão do mundo, as posições e a quantidade de elementos no mundo (permitindo também a inclusão de valores aleatórios).

```
public class Mundo02 extends br.furb.furbot.Furbot {

   @Override
   public void inteligencia(){
      if (ehVazio(ACIMA)) {
         andarAcima();
         andarDireita();
      } else {
         andarDireita();
         andarAcima();
      }
   }

   public static void main(String[] args) {
      MundoVisual.iniciar("Mundo02.xml");
   }
}
```

Figura 1 – Exemplo de código-fonte do Furbot

É possível perceber dentro da classe implementada na Figura 1 que existe um mundo visual iniciado através do método principal. Este mundo visual permite que o aluno acompanhe de forma gráfica os passos que o robô está realizando e dá controles ao aluno para gerenciar questões como velocidade de execução e poder reiniciar a execução do programa. A Figura 2a ilustra como é a interface gráfica do mundo visual do Furbot.

Quanto à evolução do *framework* para o AgenteJ, foi feita uma atualização no mundo visual, como pode-se observar na Figura2b, além de retirada a área do enunciado para que a mesma configuração de mundo possa ser reaproveitada para vários exercícios. Para a disciplina de Introdução à Programação em que foi usada para experimentar o *framework*, foi criada uma página contendo listas de exercícios e a lista de mundos. Com base no exercício a ser resolvido, o aluno estará de posse do enunciado que identifica o arquivo de mundo a ser utilizado. Além disso, foram resolvidos alguns

erros, criados métodos na classe AgenteJ para abstrair melhor algumas informações do mundo, assim como aperfeiçoados os recursos para desenvolvimento de jogos. Um dos objetivos originais do *framework* era permitir que os alunos do primeiro semestre já pudessem desenvolver pequenos jogos. Porém, ele não possuía um mecanismo de tratamento de colisões (recurso essencial em qualquer motor de jogos), levando uma certa dificuldade para os alunos tratarem desse problema. Agora a classe tem um método que é invocado toda vez que o agente colide, ou seja, encontra-se na mesma célula com outros objetos. A Figura 3 ilustra a janela do trabalho de fim do semestre: Pacman com 4 fases (cada fase é uma configuração diferente do labirinto) que podia ser resolvido em média com 320 linhas de código distribuídas entre cinco classes (cada elemento no mapa é uma classe: Pacman, Fantasma, Parede, Comida e Comida Especial).

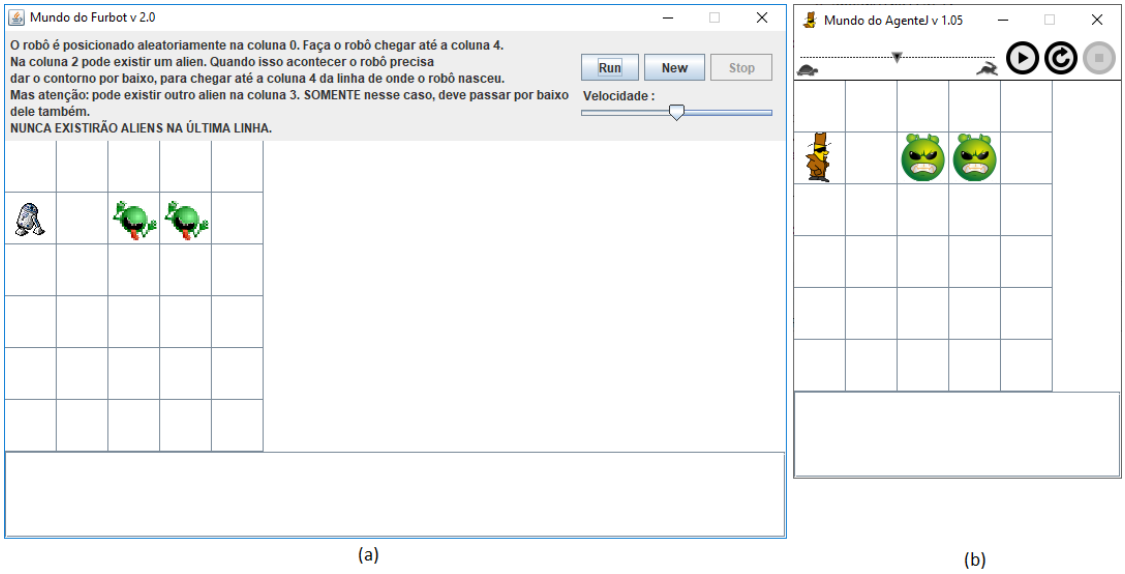


Figura 2 – Execução das classes: (a) Furbot e (b) AgenteJ

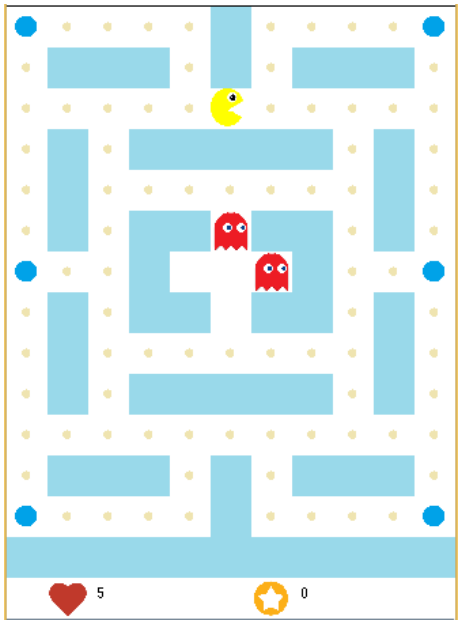


Figura 3 – Pacman com AgenteJ

Com essa reestruturação o professor já conseguia utilizar o AgenteJ em suas aulas. Entretanto, durante o semestre, foi implementado um outro recurso em que podia-se definir objetivos parciais dentro do arquivo XML, e ao executar a classe, esses objetivos eram exibidos e conferidos durante a execução. O objetivo dessa funcionalidade é permitir que o aluno identifique quais são os objetivos a serem cumpridos em cada exercício, acompanhar sua realização durante a execução da classe, e assim guiá-lo na resolução dos problemas. Por mais que o professor explique, indique, e reforce para que os alunos precisam dividir um problema em partes menores, e resolver cada uma dessas partes individualmente, os alunos acabam por tentar resolver tudo ao mesmo tempo. Então acreditava-se que com esse recurso, os alunos fossem conduzidos a dividir o problema em partes menores. A Figura 4 apresenta a nova versão do mundo visual do AgenteJ quando existem objetivos definidos no mundo. Os objetivos são listados na lateral direita da janela. Durante a execução, quando o objetivo é cumprido a caixa de verificação é marcada.

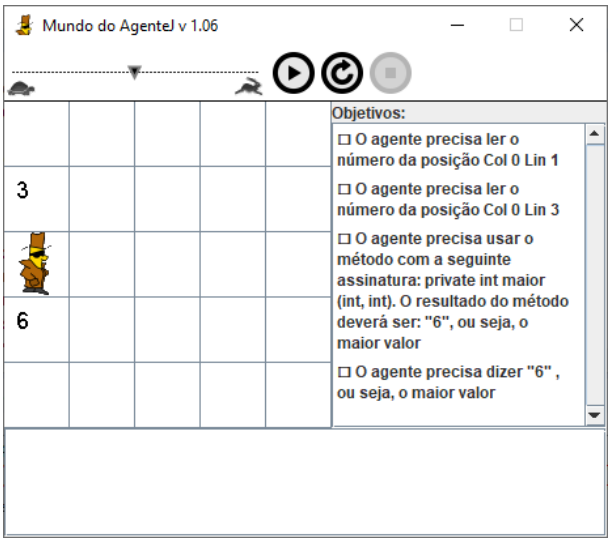


Figura 4 – Mundo Visual do AgenteJ apresentando os objetivos do exercício

A Figura 5 mostra uma solução para esse exercício. É possível notar que é um código convencional, sem que o aluno tenha que inserir marcações ou qualquer tipo de código especial para reconhecer a verificação dos objetivos. A verificação dos objetivos está integrada aos métodos do *framework*, por exemplo, dentro de *getInt* e *diga*. Os valores dos números no mundo podem ser aleatórios, inclusive para evitar que o aluno produza um código fonte específico para uma situação. Por exemplo, na Figura 4, os dois últimos objetivos estão citando o número 6, porém isso pode variar a cada execução. O único tipo de objetivo que é verificado ao final da execução é aquele em que se exige que se produza um método. Por exemplo, na Figura 4, pediu-se a criação do método "int maior(int, int)". Nesse caso a primeira verificação é a existência desse método através da API de Reflexão do Java (MCCLUSKEY, 1998). Além disso, para esse tipo de verificação foi implementado um mecanismo que altera o *bytecode* gerado pelo compilador para adicionar um código ao final do método, para guardar o resultado retornado por esse método e contar quantas vezes ele foi invocado. Esse mecanismo foi implementado usando a classe HotSwapper da biblioteca Javaassist (CHIBA, 2000). Por

isso, para executar a classe implementada com o exercício precisa-se passar os seguintes parâmetros na JVM:

```
-agentlib:jdwp=transport=dt_socket, server=y, suspend=n, address=8000
```

```
public class Ex01 extends AgenteJ {

    @Override
    public void inteligencia() throws Exception {

        andarAcima();
        int n = getInt();
        andarAbaixo();
        diga(maior(n, getInt(ABAIXO)));
    }

    private int maior(int a, int b) {
        if (a > b) {
            return a;
        } else {
            if (a < b) {
                return b;
            } else {
                return a;
            }
        }
    }

    public static void main(String[] args) {
        MundoVisual.iniciar("Mundo70.xml", Ex01.class);
    }

}
```

Figura 5 – Código da solução do exercício da Figura 4

3. Metodologia

Como o recurso de validação de objetivos foi uma inovação fornecida com o semestre em andamento, os investigadores conduziram um experimento com os alunos para avaliar se o mesmo consegue melhorar o desempenho deles durante a resolução dos exercícios. Esse desempenho foi medido através da quantidade de exercícios resolvidos durante a sessão. A hipótese a ser testada compara essa quantidade de exercícios.

H_0: $\mu_A = \mu_B$ – Não existe diferença significativa entre a quantidade de exercícios respondidos pelos participantes do Grupo A (Experimental) e do Grupo B (Controle)

O experimento foi conduzido durante as aulas de laboratório que eram dedicadas à resolução das listas de exercícios fornecidas pelo professor. Essas aulas eram organizadas a noite toda, ocupando 4 aulas (3 horas e 20 minutos). O assunto que estava sendo tratado na época era sobre métodos. A lista de exercícios para aula foi fornecida normalmente. Explicou-se à turma sobre a inovação e quem gostaria de ser monitorado para validar o recurso. Os alunos voluntariamente podiam se oferecer para o experimento.

A distribuição dos alunos foi feita de forma estratificada, a fim de construir dois grupos, um grupo experimental (Grupo A) e um grupo de controle (Grupo B). Para isso, anteriormente à aula, todos os acadêmicos foram classificados pelo professor em dois *ranks* de acordo com o seu desempenho na disciplina até aquele momento. Em seguida, criou-se um algoritmo que fazia a distribuição dos alunos de forma uniforme, ou seja, seu objetivo era construir dois grupos com a mesma quantidade de alunos com porções equivalentes de cada *rank*. Como os alunos foram convidados na própria aula, então 16

alunos aceitaram participar do experimento. Esse algoritmo foi executado na aula sobre essa amostra dos 16 alunos voluntários e a Tabela 1 apresenta a composição de cada grupo.

Tabela 1. Distribuição dos Participantes

	Grupo Experimental (A)	Grupo de Controle (B)
Rank A	6	5
Rank B	2	3
Total	8	8

O Grupo A resolveu os exercícios com o novo recurso e o Grupo B da forma que vinha resolvendo anteriormente durante o semestre. A lista continha 9 exercícios sobre métodos. Um dos investigadores era chamado para conferir e anotar quando um dos 16 alunos solucionava um exercício.

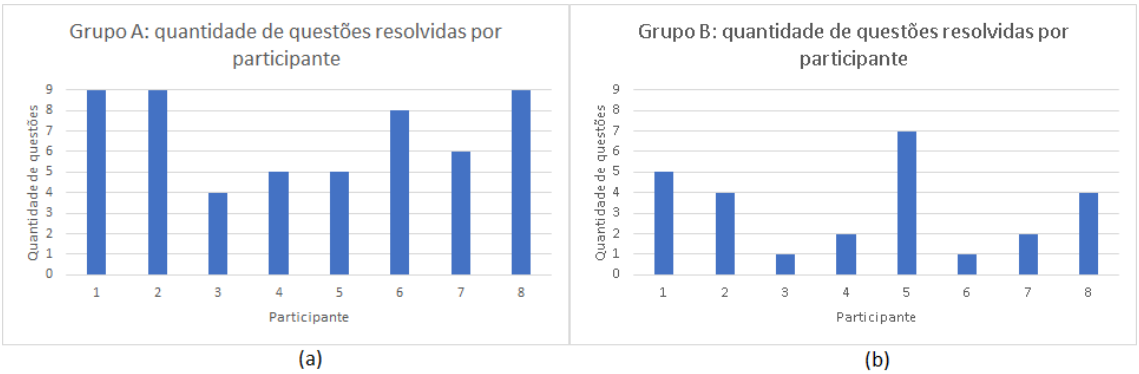
Após o encerramento da aula, os alunos do Grupo A puderam responder um inquérito contendo 5 questões para avaliarem a experiência e opinarem a respeito da nova funcionalidade. O Quadro 1 relaciona as questões desse inquérito. As questões 1 e 2 são do tipo *likert* com 5 escalas, de Concordo Totalmente a Discordo Totalmente. As demais são questões abertas.

Quadro 1. Questionário de Avaliação da Experiência

#	Descrição
1	Você acredita que o novo recurso facilita a resolução dos exercícios?
2	O seu desempenho na resolução dos exercícios foi facilitado/melhorado com o novo recurso?
3	Explique como você fazia para verificar se a solução cumpria os objetivos dos exercícios.
4	Na sua opinião qual é o maior benefício deste novo recurso?
5	Deixe um comentário ou sugestão desta nova versão do AgenteJ.

4. Resultados

A Figura 6a apresenta a quantidade de exercícios de cada aluno do Grupo A e a Figura 6b a quantidade de cada aluno do Grupo B. A Tabela 2 apresenta o resumo desses dados.



(a) (b)

Figura 6 – Quantidade de questões resolvidas por participante entre o Grupo A e o Grupo B

Tabela 2. Resumo das quantidades de exercícios resolvidos

	Total das questões	Total resolvido	Média	Mediana	Desvio padrão	Variância
Grupo A	72	55	6,875	7	2,10	4,41
Grupo B	72	26	3,25	3	2,12	4,50

Para verificar se existe diferença significativa entre os dois grupos, podia-se utilizar o Teste para Comparação de Duas Médias Populacionais. Porém, é preciso fazer um Teste para Igualdade de Duas Variâncias Populacionais utilizando a distribuição de Fisher-Snedecor para identificar se as amostras são dependentes ou independentes. Como F_{crit} (3,7870) > F_{calc} (1,0202), as duas amostras possuem variâncias amostrais equivalentes, concluindo que é possível realizar o Teste para Igualdade de Duas Variâncias Populacionais com Amostras Independentes. Em um nível de significância de $\alpha = 0,05$, o resultado do teste demonstrou que $|t_{calc}| > t_{crit}$ com t_{calc}=-3,4348 e t_{crit}=1,7613, rejeitando H_0. Conforme pode-se observar na Tabela 2, μA> μB indicando que o Grupo que usou o recurso resolveu, com significância estatística, mais exercícios que o Grupo que não usou o recurso.

Sete alunos do Grupo A participaram da avaliação qualitativa da experiência. Todos os alunos responderam "*Concordam*", seja totalmente ou parcialmente, para as questões 1 e 2. Os alunos responderam que a forma que eles faziam para verificar se os objetivos do exercício eram atendidos (questão 3) foi através de testes de mesa, depuração ou pela observação do comportamento do agente no mundo. A questão 4 solicitava uma opinião sobre o maior benefício com o recurso. O Quadro 2 apresenta as respostas, e ficou evidente que os objetivos expostos do exercício serviram para conduzir a solução do exercício.

Quadro 2. Respostas sobre o maior benefício do novo recurso

"Já temos a resposta dos exercícios, só precisamos projetar o código para chegar até ela"
"Dar direcionamento às pessoas que não possuem domínio em lógica de programação"
"Melhor resolução de erros"
"Facilidade de compreender o que o exercício está pedindo, e a possibilidade de verificar se alcançou todos os objetivos propostos de maneira mais rápida"
"Visualização melhor dos objetivos à serem alcançados"
"Sentimento de progresso"
"Agilidade na verificação das etapas do código"

A questão 5 solicitava uma opinião para melhorar o recurso. O Quadro 3 apresenta as respostas e não houveram sugestões, mas ficou claro que os alunos gostariam de ter esse recurso desde o início do semestre.

Quadro 3. Opiniões e sugestões de melhorias

"Top"
"Gostei e acredito que será muito útil se utilizado desde as primeiras aulas do semestre"
"Show"
"Nada"
"Achei muito bom, facilita no desenvolvimento do problema para alcançar seus objetivos com uma visão facilitada"
"Espero ter nos próximos exercícios, ajudou bastante"
"Gostei bastante da ferramenta, apesar de ter feito a verificação somente no final, assim concluindo o código, porém auxilia muito em exercícios mais complexos. Talvez se tivesse mais tempo de uso, poderia dar alguma sugestão."

5. Considerações Finais

O AgenteJ permite que o aluno desenvolva algoritmos da linguagem Java e execute-os com apoio de uma interface gráfica para melhor entendimento de conceitos abstratos de programação. Por isso, pode ser considerado como sendo um ambiente facilitador do aprendizado. Entretanto, ele não disponibilizava nenhum recurso que fornecesse um retorno imediato a respeito das soluções dos exercícios desenvolvidos pelos alunos. Esse trabalho apresentou um verificador automático de soluções que acompanhava a execução da classe. Através de um experimento com 16 alunos foi possível conferir com significância estatística que os alunos que usaram o verificador resolveram mais exercícios. O questionário de avaliação da experiência dos alunos ajudou a identificar uma grande aceitação do verificador por parte deles, apontando uma série de benefícios como o "sentimento de progresso" e uma "melhor visualização dos objetivos à (sic) serem alcançados". Os alunos admitiram que o recurso de verificação de objetivos conduziu a tarefa deles de solucionar o problema demonstrando ser uma ferramenta útil para o aprendizado de programação.

Em relação a trabalhos similares apontados na introdução, o recurso apresentado nesse artigo diferencia-se por dar suporte ao aluno durante a resolução do problema sem a intervenção do professor, através de uma sistemática de verificação implementado pelo uso ou não de determinado método do *framework*. Esse trabalho não defende a ausência do professor no processo ensino-aprendizagem, mas segue o princípio de que o aluno tente primeiro por si, antes de solicitar ajuda ao professor.

O download do AgenteJ, o manual do usuário e as listas de exercícios está disponível no endereço https://sites.google.com/site/adilsonv77/agentej.

Agradecimentos

Os autores agradecem aos alunos e professor da disciplina de Introdução à Programação do curso de Bacharelado em Engenharia de Software da Universidade do Estado de Santa Catarina pela disponibilidade na realização dos testes.

Referências

ARAÚJO, L.; SILVEIRA, H. U. C. DA; MATTOS, M. **Ensino do pensamento computacional em escola pública por meio de uma plataforma lúdica**. Anais Workshops do VII Congresso Brasileiro de Informática na Educação. **Anais**...Fortaleza, CE: 2018.

BARBOSA, A. A. et al. **Uso de algoritmos de similaridade para classificar códigos de acordo com a taxonomia SOLO em disciplinas de programação introdutória**. Anais dos Workshops do V Congresso Brasileiro de Informática na Educação (CBIE 2016). **Anais**...Uberlandia, MG: 2016.

BERGIN, J. et al. **Patterns for Experiential Learning**. 6th European Conference on Pattern Languages of Programms. **Anais**...Irsee, Germany: 2001.

CHIBA, S. Load-Time Structural Reflection in Java. In: SPRINGER VERLAG (Ed.). . **ECOOP 2000 - Object-Oriented Programming, LNCS 1850**. [s.l: s.n.]. p. 313–336.

ESTRÁZULAS, D. S. et al. **Ensinando Programação Através de Dispositivos Móveis: Mobile Furbot e iFurbot**. XVIII Seminário de Computação - FURB. **Anais**...Blumenau, SC: 2009.

GUZDIAL, M.; SOLOWAY, E. Teaching the Nintendo generation to program. **Communications of the ACM**, v. 45, n. 4, p. 17, 2002.

KUTZKE, A. R.; DIRENE, A. **Mediação do erro na educação: um arcabouço de sistema para a instrumentalização de professores e alunos**. Anais do XXV Simpósio Brasileiro de Informática na Educação (SBIE). **Anais**...Dourados, MS: 2014.

LOPES, P. P.; GOMES, M. S.; DANTAS, T. F. **Proposta de um Sistema para o Monitoramento das Atividades de Programação Para Alunos Iniciantes**. Anais dos Workshops do VI Congresso Brasileiro de Informática na Educação. **Anais**...Recife, PE: 2017.

MCCLUSKEY, G. **Using Java Reflection**. 1998. Disponível em: <https://www.oracle.com/technetwork/articles/java/javareflection-1536171.html>.

ROBINS, A.; ROUNTREE, J.; ROUNTREE, N. Learning and teaching programming: A review and discussion. **Computer Science Education**, v. 13, n. 2, p. 137–172, 2003.

SANTOS, F. A. O.; SEGUNDO, P. S. C.; TELVINA, M. S. **CodeTeacher : Uma Ferramenta para Correção Automática de Trabalhos Acadêmicos de Programação em Java**. Anais dos Workshops do VI Congresso Brasileiro de Informática na Educação. **Anais**...Recife, PE: 2017.

VAHLDICK, A.; MATTOS, M. **Relato de uma Experiência no Ensino de Algoritmos e Programaçao Utilizando um Framework Lúdico**. XIX Simpósio Brasileiro de Informática na Educação. **Anais**...Fortaleza, CE: 2008.

WINSLOW, L. E. Programming pedagogy: A psychological overview. **ACM SIGCSE Bulletin**, v. 28, n. 3, p. 17–22, 1996.