CENTRO UNIVERSITÁRIO UNIVATES
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

ESTENDENDO OS RECURSOS DO GERADOR AUTOMÁTICO DE EXERCÍCIOS PARA A APRENDIZAGEM DOS CONCEITOS BÁSICOS DE ALGORITMOS E PROGRAMAÇÃO

TAILOR PIUCO

Lajeado, dezembro de 2016

TAILOR PIUCO

ESTENDENDO OS RECURSOS DO GERADOR AUTOMÁTICO DE EXERCÍCIOS PARA A APRENDIZAGEM DOS CONCEITOS BÁSICOS DE ALGORITMOS E PROGRAMAÇÃO

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Tecnológicas do Centro Universitário UNIVATES, como parte dos requisitos para a obtenção do título de bacharel em Engenharia da Computação.
Área de concentração: Algoritmos e Programação

ORIENTADOR: Juliano Dertzbacher

Lajeado, dezembro de 2016

AGRADECIMENTOS

Muitas foram as pessoas que participaram desta caminhada, e a elas gostaria de agradecer.

Agradeço aos meus pais Jones Piuco e Inês Piuco pelo incentivo, pelo amor incondicional e pelos seus esforços para que esta etapa fosse concluída. Agradeço seus exemplos, que foram essenciais para o ser humano que me tornei. Agradeço aos meus irmãos pelo incentivo e pelas “brincadeiras” para que este momento chegasse logo.

Agradeço a minha namorada pela compreensão e auxílio na elaboração deste trabalho.

Agradeço ao meu orientador Juliano Dertzbacher pelos momentos de orientação e dedicação para a elaboração deste trabalho.

Agradeço ao coordenador do curso Marcelo Malheiros pelos momentos de auxílio e compartilhamento do conhecimento para a elaboração da parte prática deste trabalho.

Agradeço a todos os professores que se dispuseram a testar e avaliar a ferramenta e, que tão importante foi para a avaliação e conclusão deste trabalho.

Agradeço a todos os professores que ao longo da graduação se dedicaram a ensinar, transferindo seu conhecimento que, de alguma forma, auxiliou no desenvolvimento deste trabalho.

Agradeço a todos os amigos que, de uma forma ou de outra, ajudaram e torceram para que este momento chegasse.

RESUMO

Um dos maiores desafios dos professores das disciplinas de Algoritmos e Programação é fazer com que o conteúdo da disciplina seja assimilado de forma mais eficiente. Várias ferramentas foram desenvolvidas para auxiliar os professores e alunos no processo de aprendizado. Contudo, estas disponibilizam somente listas estáticas de exercícios. Ciente desta limitação, foi desenvolvido um gerador automático de exercícios, que permite ao aluno ou professor elaborar exercícios em uma linguagem, baseada em pseudocódigo e executar o sistema gerador para criar o exercício com a solução em *Java*, *C*, *C++* e *Python*. Este trabalho tem como objetivo estender os recursos básicos do sistema gerador original, utilizando variáveis estruturadas, como vetores e matrizes, funções e procedimentos. Para que este objetivo seja atingido, serão feitas alterações pontuais no sistema gerador, de modo que contemple todos os conteúdos estudados na disciplina de Algoritmos e Programação. O sistema gerador tem a premissa de auxiliar os professores na aplicação de exercícios aos alunos e servir de estudo extraclasse para os alunos, para que eles tenham uma melhor concepção do conteúdo. A coleta e análise dos resultados foram realizadas a partir de uma pesquisa quantitativa aplicada aos professores. Eles tiveram a oportunidade de realizarem testes em um protótipo da ferramenta e tiveram a oportunidade de elaborar os *templates* e gerar exercícios. Com o desenvolvimento do sistema, pretende-se oferecer uma experiência ampliada, qualificando a aprendizagem da disciplina de Algoritmos e Programação, e também melhorar o aproveitamento dos alunos nas disciplinas que a sucedem e utilizam os fundamentos desta como a base.

Palavras-chave: Algoritmos e Programação, gerador de exercícios, ensino de programação, pseudocódigo.

ABSTRACT

One of the greatest challenges for the teachers in the Algorithms and Programming disciplines is to make the content of the course be assimilated more efficiently. Several tools have been developed to assist teachers and students in the learning process, however, they only provide static exercise lists. Aware of this limitation, an automatic exercise generator was developed that allows the student or teacher to elaborate exercises in a pseudocode-based language and execute the generator system to create the exercise with the solution in *Java*, *C*, *C++* and *Python*. This work aims to extend the basic features of the original generator system using structured variables such as arrays and matrices, functions and procedures. In order for this objective to be achieved, specific changes will be made to the generating system, so that it will contemplate all the contents studied in the Algorithms and Programming discipline. The generator system has the premise of assisting teachers in the application of exercises to students and serve as an extraclass study for students, so that they have a better conception of the content. The results were collected and analyzed based on a quantitative research applied to teachers. They had the opportunity to test a prototype of the tool and had the opportunity to elaborate the templates and generate exercises. With the development of the system, it is intended to offer an extended experience, qualifying the learning of the Algorithms and Programming discipline, and also to improve the students' achievement in the disciplines that succeed and use the fundamentals of this as the basis.

Keywords: Algorithms and Programming, exercise generator, programming teaching, pseudocode.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o passar do tempo, várias ferramentas surgiram para ajudar no ensino de programação nas disciplinas iniciais dos cursos de informática. Mesmo com diversas ferramentas disponíveis como o *VisuAlg* (VISUALG, 2016), o *Scratch* (SCRATCH, 2016), o *Learneroo* (LEARNEROO, 2016), entre outras, ainda são raras as que possuem recursos de geração automática de exercícios. As ferramentas citadas e outras trabalham com uma lista estática. Com isso, surge a necessidade de recursos que ofereçam exercícios diversificados para serem aplicados aos alunos. Ciente desse fato, foi desenvolvido um gerador automático de exercícios, que tem por objetivo diversificar os exercícios propostos, não ficando limitado a uma lista estática.

O gerador foi desenvolvido pelo aluno Pedro Tramontina, do curso de Engenharia da Computação da UNIVATES, no ano de 2015, como trabalho de conclusão de curso. Os exercícios compreendem as linguagens *C*, *C*++, *Java* e *Python*. A ferramenta é baseada em *templates,* conceito que diversifica os exercícios que podem ser aplicados junto aos alunos. O *template* é um arquivo texto escrito pelo professor, que contém o enunciado do exercício, a solução em formato de pseudocódigo e algumas dicas. O gerador interpreta o *template,* altera os valores das variáveis para algum valor aleatório ou um valor pré-estabelecido pelo professor e exibe ao aluno para que seja resolvido.

Contudo, uma quantidade limitada de comandos foi implementada, como por exemplo, o comando *if* e *while*. Desta forma, mesmo gerando exercícios aleatórios, o gerador fica restrito e não contempla todo o conteúdo didático apresentado nas disciplinas de algoritmos e programação.

1.1 Objetivo geral

O principal objetivo deste trabalho é incorporar ao gerador automático de exercícios de programação novos recursos, para que os alunos tenham uma evolução significativa na aprendizagem, reforçando todo o conteúdo estudado em aula.

1.2 Objetivos específicos

- Aprimorar o pseudocódigo para que suporte exercícios com o laço de repetição *do ... while*.
- Possibilitar a utilização da abordagem de vetores e matrizes.
- Oferecer recursos para o desenvolvimento de procedimentos e funções.
- Ampliar o conjunto de operações matemáticas.
- Implementar um dicionário para tradução dos exercícios para o português.
- Avaliar a ferramenta junto aos professores da disciplina de Algoritmos e Programação.

Este trabalho é composto por 7 capítulos, incluindo a introdução. O Capítulo 2 é o referencial teórico, o qual tem o objetivo de apresentar as diversas ferramentas para auxílio do ensino de programação disponíveis no mercado. No Capítulo 3, é apresentada a metodologia, onde é feito um estudo dos recursos já existentes na ferramenta e os que serão estendidos neste trabalho. No Capítulo 4 é apresentada uma visão geral mais detalhada dos comandos e funcionalidades estendidas do Gerador Automático de Exercícios. No Capítulo 5, é feito o detalhamento do desenvolvimento dos comandos a serem estendidos no sistema gerador. No Capítulo 6, pode ser observada a avaliação da ferramenta com os professores e também é apresentada uma análise sobre os resultados obtidos com os testes. Por fim, no Capítulo 7 são feitas as considerações finais sobre o presente trabalho, como também uma análise sobre futuros trabalhos na área.

2 REVISÃO BIBLIOGRÁFICA

Neste Capítulo serão abordados tópicos que irão embasar teoricamente o trabalho proposto e demonstrarão a importância da disciplina de algoritmos e programação nos cursos de informática, servindo de base para outras disciplinas lógicas. Afinal, algumas técnicas estão sendo implementadas e aprimoradas para ajudar os alunos e professores na concepção do conteúdo, com novas formas de apresentá-lo e aplicar exercícios.

Nos próximos tópicos a disciplina de algoritmos e programação será contextualizada, assim como o método de ensino, técnicas e ferramentas utilizadas por alunos e professores para auxiliá-los neste processo.

2.1 Ensino tradicional de programação

Nos últimos anos, a demanda por profissionais nas áreas de engenharia e informática tem aumentado e, por consequência, a procura pelos cursos relacionados a estas áreas também. Independentemente das instituições, o aluno terá de passar pelas mesmas disciplinas introdutórias e, neste contexto, a disciplina de algoritmos e programação é clássica.

Normalmente, o método utilizado nestas disciplinas introdutórias é o tradicional. Inacio (2014) diz que o método tradicional é constituído, inicialmente, de uma apresentação da estrutura do computador e seu funcionamento básico. Após esta etapa inicial, é apresentado aos alunos o conceito de fluxogramas e, a partir desta abordagem, eles terão uma primeira ideia da estrutura de um programa de computador. Nesta etapa, pode-se utilizar de ferramentas visuais como o *VisuAlg* (VISUALG, 2016) ou *Scratch* (SCRATCH, 2016) para uma melhor assimilação do conteúdo. Para finalizar, os alunos fazem as atividades práticas com uma linguagem

de programação estabelecida, como *Java*, *C*, *Phyton* ou outra. O método se baseia na ideia de que quanto mais o aluno praticar, mais vai assimilar o conteúdo e, desta forma, esta tarefa se torna cansativa, pelo fato dos exercícios serem repetitivos em demasia.

O método tradicional de ensino de programação, segundo Ferradin e Stephani (2005), pode ser visto como uma apresentação da teoria de forma simplificada, exemplos básicos de lógica, seguidos de exercícios que vão aumentando de dificuldade, de acordo com a evolução dos alunos perante o conteúdo.

Complementando, Souza (2009) nos diz que a disciplina de algoritmos e programação não tem início na apresentação da linguagem de programação. A disciplina consiste em embasamento sólido em lógica e, para isso, são aplicados exercícios, nos quais o aluno utiliza pseudocódigos para resolvê-los.

De acordo com Rocha (2010), existem alguns fatores que dificultam o processo de aprendizagem como a exigência de lógica e matemática predominante em toda a disciplina, dificuldades em captar o conteúdo por parte dos alunos e também o ritmo de aprendizagem de cada um, resultando em um grande número de desistências ou troca de curso. O bom entendimento do conteúdo nestas disciplinas iniciais é crucial para o desenvolvimento do aluno e o bom rendimento nas disciplinas mais avançadas do curso. Para o professor, é muito difícil identificar as dificuldades de cada um e, com isso, podem ocorrer falhas no processo de aprendizagem.

Radošević (2010) aponta algumas razões pelas quais a disciplina se torna difícil: a falta de conhecimento prévio em informática e matemática, habilidades pouco desenvolvidas sobre pensamento abstrato e raciocínio lógico, a falta de motivação e o medo de programação.

Assim, percebe-se que o método tradicional de ensino de algoritmos e programação pode ser aperfeiçoado. Independentemente do nível de experiência do aluno ou da facilidade com conceitos de matemática, o estudo extraclasse é fundamental para a compreensão dos recursos e estratégias apresentadas na disciplina. Visando melhorar o rendimento do aluno, algumas técnicas foram implementadas para que este possa compreender melhor o conteúdo, podendo assim, dedicar-se também ao estudo extraclasse.

2.2 A disciplina de programação

A disciplina de programação pode ser resumida em aulas expositivas e práticas, nas quais o andamento e a velocidade que o conteúdo é apresentado aos alunos parte do professor. Pode ser que o ritmo empregado pelo professor seja muito rápido, prejudicando a compreensão do conteúdo; ou o desenvolvimento da disciplina ocorre de forma muito lenta, pois alunos com alguma experiência prévia estarão revendo o que está sendo apresentado, fazendo com que fiquem dispersos e passem a não contribuir de forma significativa nas aulas (ROCHA, 2010).

Os conteúdos abordados na disciplina se referem aos conceitos fundamentais, representação de dados, operações, instruções de entrada e saída, desvio condicional, laços de repetição, tipos compostos como vetores e matrizes, modularização e exercícios, utilizando alguma linguagem de programação (DETERS 2008). Sobre a abordagem da linguagem de programação, Deters (2008) destaca que:

> Ainda, dependendo da fase em que a disciplina é lecionada, o ensino pode estar atrelado à utilização de alguma linguagem de programação para a construção de programas. Entretanto, não existe um padrão definido, cabendo a cada Instituição de Ensino (IES) selecionar e organizar a estrutura de seu curso, conforme a necessidade de ensino ou de mercado.

No decorrer da disciplina, alguns alunos podem demonstrar mais dificuldades de aprendizagem do que outros. Isso pode ser atribuído à capacidade de aprendizagem de cada um ou, também, às experiências prévias vividas em ambiente de trabalho ou autoestudo.

Para o professor, esta disparidade deve ser tratada para que no final todos os alunos assimilem o conteúdo de maneira satisfatória e, para que este objetivo seja alcançado, vários desafios devem ser superados, como exposto a seguir.

2.3 Desafios no ensino de programação

De acordo com Deters (2008), há três grandes desafios a serem superados nas disciplinas de programação, que são: (i) o grande número de alunos na mesma sala de aula, que faz com que o professor não consiga atender dúvidas individuais dos alunos; (ii) as avaliações somente por meio de provas escritas ou trabalhos

individuais, não promovendo uma aprendizagem satisfatória; e (iii) a disparidade de conhecimento, experiência e ritmo de aprendizagem por parte dos alunos.

Outro problema que deve ser ressaltado é o fato dos alunos não terem noção da importância das disciplinas introdutórias de programação para o restante do curso e formação dos mesmos. Este desconhecimento é mais comum em alunos sem experiência prévia, pois não conseguem aplicar os conteúdos na prática. Algumas dificuldades também são vivenciadas pelos professores, conforme aponta Deters (2008):

> (i) a capacidade de reconhecer habilidades inatas de seus alunos; (ii) a apresentação de técnicas de resolução de problemas; (iii) como promover o desenvolvimento da capacidade de abstração do aluno, permitindo-o selecionar as estruturas de dados coerentes; (iv) como facilitar a cooperação e colaboração entre os alunos.

O professor ainda deve auxiliar seus alunos a elaborar a melhor estratégia para resolver os problemas propostos em aula, utilizando algoritmos, pois cada aluno deverá aprender qual é o seu estilo de raciocinar na resolução de problemas. O aluno terá que codificar o algoritmo em uma linguagem de programação pré-estabelecida pela Instituição de Ensino, podendo assim, mensurar a complexidade da solução elaborada e tendo por fim, alcançado os objetivos das disciplinas de algoritmos e programação (GIRAFFA, 2003).

Um dos maiores desafios para os professores é fazer uma leitura do nível de conhecimento da turma. Uma leitura bem feita fará com que o professor saiba qual é a necessidade de cada aluno e, com isso, eles terão um melhor aproveitamento e compreensão do conteúdo. Todos os semestres, os professores têm como meta superar estes desafios. Cada aluno absorve o conhecimento de forma e ritmo diferentes.

2.4 Processo de aquisição do conhecimento

O ponto de partida para uma aprendizagem significativa são os conhecimentos prévios dos alunos, suas vivências e hábitos fora do ambiente acadêmico. Estes conhecimentos prévios devem ser integrados, a fim de serem aperfeiçoados, para a absorção do conhecimento de fato (MINGUET, 1998).

Uma técnica muito utilizada pelos professores são os testes de nivelamento ou conhecimentos prévios. Isso faz com que sejam identificadas as experiências

prévias de cada estudante, possibilitando ao professor avaliar e aplicar o conteúdo da disciplina de forma que todo aluno tenha o conhecimento independente de seus conhecimentos prévios, o que sabiamente observa Minguet (1988, p.168).

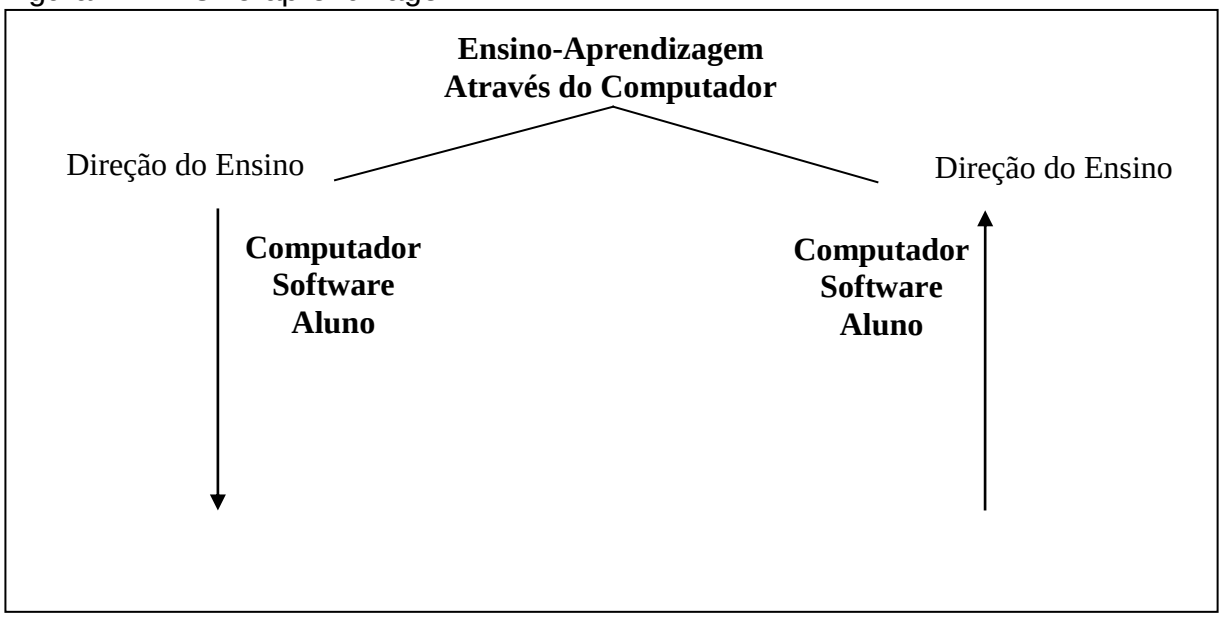
> Neste sentido, os bons professores utilizam a avaliação prévia ou avaliação inicial como instrumento para determinar os conhecimentos prévios dos alunos. Uma elaboração cuidadosa de tal avaliação pode fazer dela uma boa ferramenta desta tarefa. Ainda assim, será insuficiente se não for complementada com um trabalho contínuo de seguimento e de contraste, não se reduzindo este exercício de análise a uma tarefa pontual no começo do curso.

O conhecimento não se restringe somente ao espaço das salas de aula, ele deve ser buscado nos mais diferentes lugares. Isto reflete na capacitação e aprimoramento da força de trabalho. Estudos apontam que a produção de conhecimento aumentou em espaços cada vez menores de tempo e tende a aumentar de forma crescente. Desta forma, as pessoas precisam aprender mais em menos tempo e este ciclo se repete continuamente (PEREIRA, 2001).

O uso do computador na educação tem por objetivo auxiliar o processamento de informações e aquisição do conhecimento, não apenas assuntos relacionados à computação, como qualquer outro que haja interesse (VALENTE 1998).

O aprendizado, através do computador, implica que o aluno possa adquirir conceitos sobre qualquer domínio, porém, pela ótica pedagógica, isto ocorre de maneira variada, oscilando entre dois polos, conforme se observa na Figura 1 (VALENTE 1998).

Figura 1 – Ensino-aprendizagem



Fonte: Valente, 1998, p.2.

Como observado na Figura 1, o acadêmico que utiliza alguma linguagem de programação, está ensinando o computador. Da mesma forma, o contrário acontece, quando o aluno utiliza o computador para estudo. Neste momento, o computador assume o papel de ensinar, pois no lugar dos tradicionais livros, a ferramenta usada é o software do computador (VALENTE 1998).

Fazendo uso de ferramentas para estudo, no momento em que o aluno resolve os enigmas ou responde às perguntas propostas pela ferramenta, ele está praticando e, consequentemente, adquirindo conhecimento, ou seja, está assimilando o conteúdo.

Atualmente, são utilizadas várias ferramentas em sala de aula e também no ambiente extraclasse, como complemento do conteúdo apresentado na disciplina.

2.5 Ferramentas de aprendizagem de programação

Várias ferramentas estão disponíveis para auxiliar professores e alunos com o conteúdo das disciplinas de programação. Como exemplos, tem-se o *VisuAlg* (VISUALG, 2016), o *.Net Fiddle* (NET FIDDLE, 2016), o *Learneroo* (LEARNEROO, 2016), o *Scratch* (SCRATCH, 2016), entre outras.

Segundo Radošević (2010), estas ferramentas podem ser divididas em duas categorias: sistemas de avaliação automáticos, como por exemplo o *VisuAlg* e o *Learneroo*, e compiladores *online,* como por exemplo o *.Net Fiddle* e o *Scratch*.

Os exercícios de programação dos sistemas de avaliação automáticos incluem todos os sistemas que avaliam parte ou todos os aspectos do programa criado pelo aluno. Os sistemas mais antigos eram baseados em um método muito simples de comparação de saída. O professor criava um programa modelo, que por sua vez, gerava uma saída e era comparada com a saída do programa do aluno (RADOŠEVIĆ, 2010).

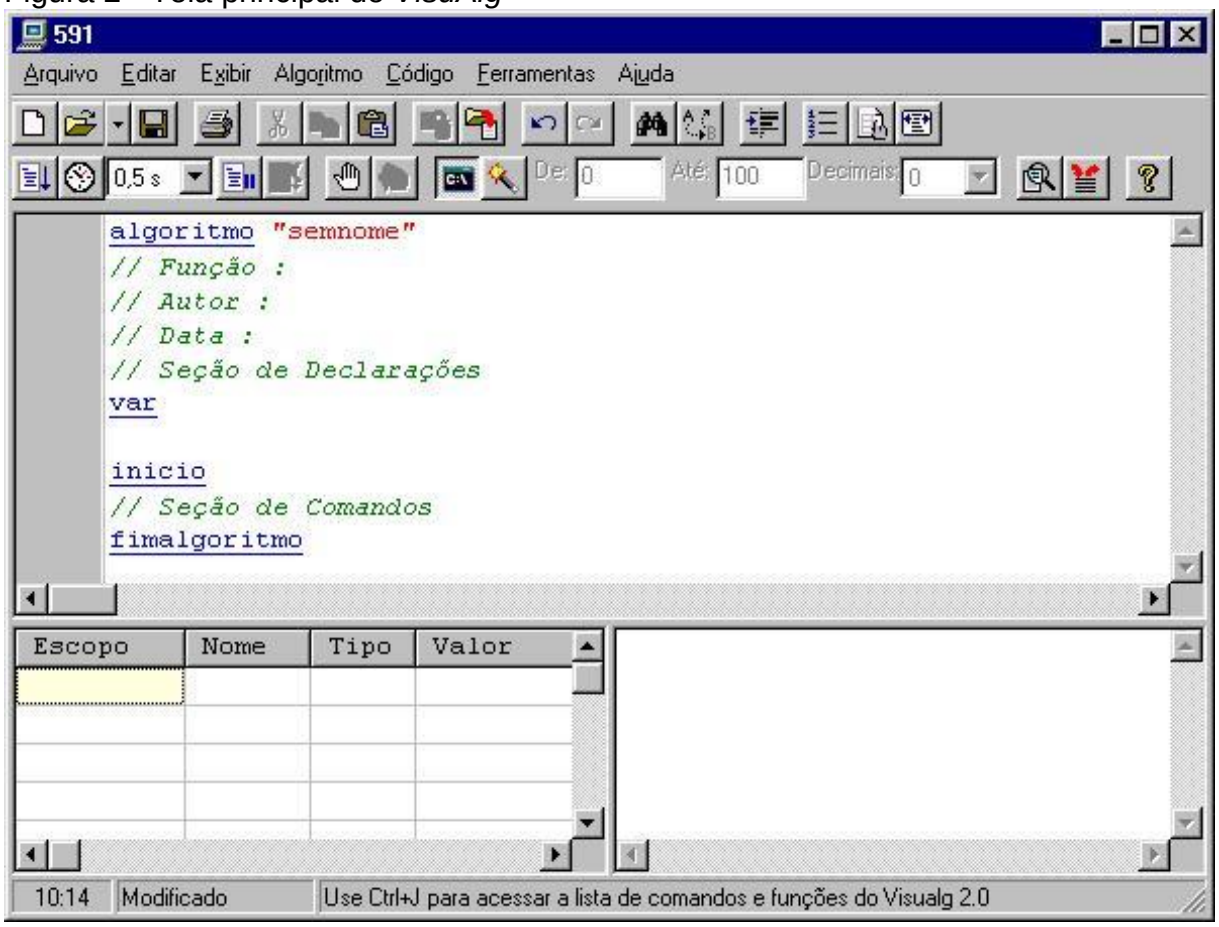
Os compiladores *online* são ferramentas que também servem como auxílio ao ensino de programação. A grande vantagem destas ferramentas é que o aluno não precisa ter o compilador instalado em sua máquina local, podendo ser acessível de qualquer lugar do mundo, via internet, por um navegador. O único pré-requisito é que a linguagem utilizada seja suportada pelo compilador (RADOŠEVIĆ, 2010).

O *VisuAlg* é uma ferramenta gráfica onde os alunos podem escrever seus programas, em formato de pseudocódigo, executar e também depurá-los (VISUALG, 2016). Conforme Souza (2009), o *VisuAlg* pode ser definido como:

> O *VisuAlg* é um aplicativo que fornece aos estudantes que se iniciam nas disciplinas de programação ferramentas para digitar, executar e depurar o pseudocódigo para resolver problemas propostos nas aulas e em exercícios, fornecendo também aos professores vários recursos didáticos para que expliquem como os programas funcionam [...]

O ambiente de desenvolvimento do *VisuAlg* é simples e intuitivo. Sua tela principal é composta da barra de tarefas, do editor de textos, do quadro de variáveis, do simulador de saída e da barra de status. Quando a tela é aberta, fica visível uma estrutura predefinida de um pseudocódigo, que tem o intuito de agilizar a escrita do código, como pode ser visto na Figura 2 (VISUALG, 2016).

Figura 2 - Tela principal do *VisuAlg*



Fonte: VisuAlg, 2016.

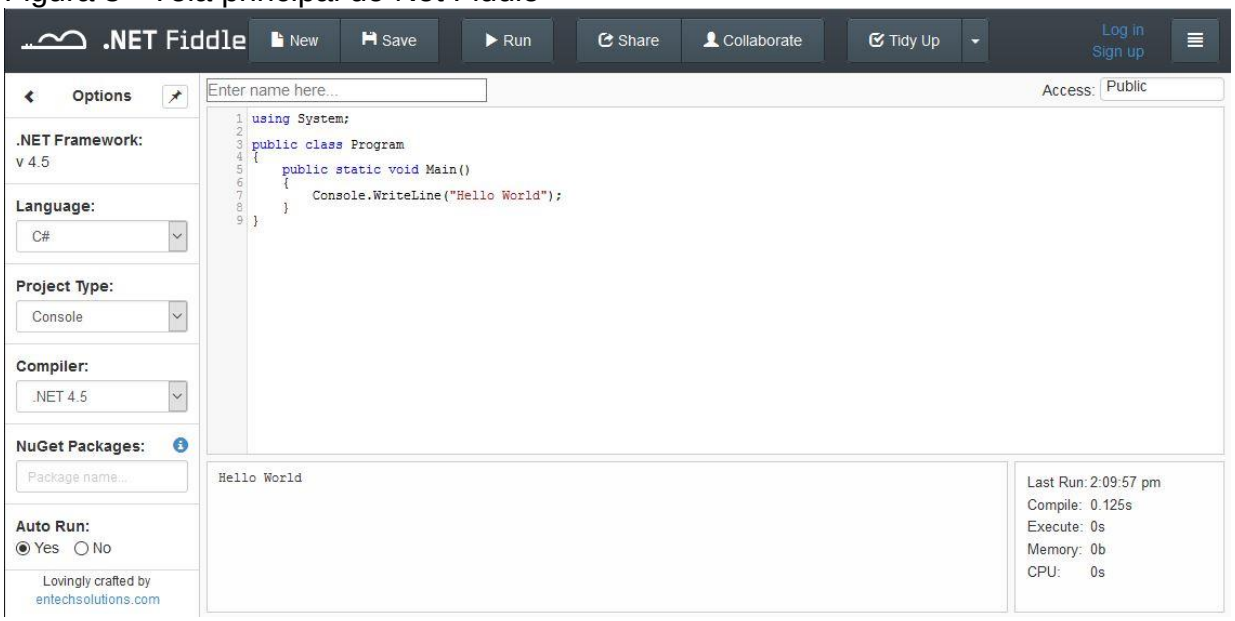
O *VisuAlg* utiliza como linguagem o *Portugol*, que é bastante utilizado em disciplinas de algoritmos e programação em cursos de todo o Brasil. Sua sintaxe se assemelha ao *Pascal*, sem utilizar o ponto e vírgula para separar os comandos (SOUZA, 2009), (VISUALG, 2016).

O *VisuAlg* se destaca pela linguagem utilizada, o *Portugol*, que é de fácil compreensão para iniciantes na área da programação. Com a ferramenta é possível simular diversos tipos de programa. Do mesmo modo que a linguagem de fácil acesso é um diferencial, ela se torna um limitador quando o aluno deseja aprofundar seus conhecimentos em alguma linguagem específica. Outra desvantagem do *VisuAlg* é que a ferramenta roda somente em sistemas operacionais *Windows* da *Microsoft*.

O *.Net Fiddle* é um compilador *online,* no qual podem ser escritos códigos para a plataforma *NET* nas linguagens *C#*, *VB.NET* e *F#*. É uma ferramenta muito boa para o aprendizado, pois o aluno não precisa ter instalado em sua máquina local

o pacote do *Net Framework*, que é necessário para compilar e executar programas desta plataforma. Conta com uma interface simples, visando à praticidade de seu uso, como pode ser visto na Figura 3 (NET FIDDLE, 2016).

Figura 3 - Tela principal do Net Fiddle



Fonte: Net Fiddle, 2016.

O *Net Fiddle* se enquadra no grupo dos compiladores *online*, que permite a escrita de qualquer tipo de código, seguindo a sintaxe das linguagens de programação preestabelecidas: *VB.NET, C# e F#*. Algumas vantagens desta ferramenta são: ambiente totalmente online, suporte a várias linguagens de programação e possibilidade de escolha entre diferentes versões do *Net Framework*. O ponto negativo está no fato de que o aluno só poderá aplicar os conhecimentos adquiridos com a ferramenta em um ambiente com o sistema operacional *Windows*, pois a ela foi projetada para suportar somente programas que são executados na plataforma *NET* da *Microsoft*.

O *Scratch* foi desenvolvido por um grupo de pesquisa do *MIT (Massachusetts Institute of Technology) Media Lab* chamado *Lifelong Kindergarten Group (LLK),* com o intuito de facilitar e agilizar a introdução dos alunos ao ambiente de programação, mesmo que estes não possuam nenhuma experiência prévia. O ambiente de programação do *Scratch* possui alguns objetos nos quais os alunos atribuem funções para dar vida a eles, conforme pode ser observado na Figura 4 (AURELIANO, 2012).

Figura 4 - Ambiente de desenvolvimento do Scratch

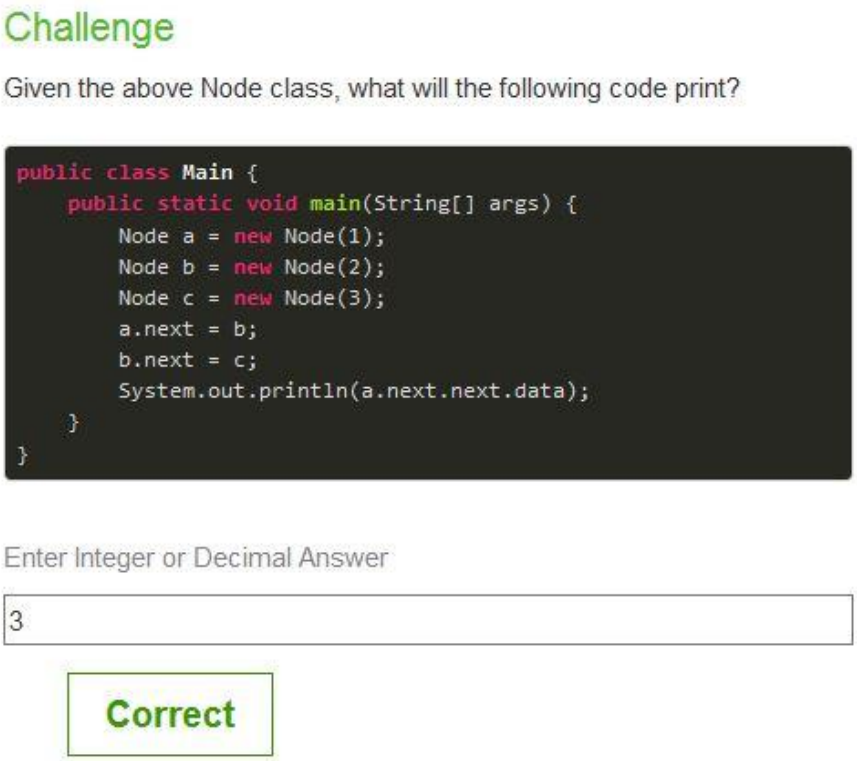
Fonte: Aureliano, 2012.

O *Scratch* utiliza uma linguagem de programação visual, permitindo que o aluno manipule imagens e músicas, criando histórias interativas. Para dar dinâmica às histórias, o aluno precisa vincular imagens, músicas ou variáveis aos objetos disponíveis no *Scratch*. Os objetos são colocados na história pelo método de arrastar e soltar (AURELIANO, 2012).

O *Scratch* é uma ferramenta que utiliza uma linguagem própria, com o objetivo de ser de fácil compreensão. Ela é indicada para crianças a partir dos oito anos de idade. Sua interface amigável e dinâmica de clicar e arrastar os comandos é seu grande diferencial, além de ser uma ferramenta *online*. A desvantagem é que o aluno terá que utilizar outras ferramentas para aprofundar seus estudos, pois com os comandos disponíveis, somente trechos básicos podem ser construídos.

O *Learneroo* é uma ferramenta *online,* com a qual é possível aprender vários conteúdos pertinentes à programação como algoritmos, desenvolvimento para *web*, conceitos de matemática, *Java* e práticas de programação. Cada assunto pode ser estudado e, no decorrer da explicação, alguns desafios são propostos para o aluno, como pode ser visto na Figura 5 (LEARNEROO*,* 2016).

Figura 5 - Learneroo. Desafio em algoritmos



Fonte: Learneroo, 2016.

Na Figura 5, pode ser observado que na apresentação do conceito de nós, ou *nodes* em inglês, foi proposto um desafio ao aluno, onde é exibida uma situação, na qual alguns nós são criados e valores são atribuídos a eles. Com estas informações o aluno tem que analisar o código e responder a pergunta proposta: “Dada a classe *Node* abaixo, o que o código a seguir irá imprimir?”. Quando o aluno submeter sua resposta, aparecerá se está correta ou errada (LEARNEROO, 2016).

O *Learneroo* é uma ferramenta que possibilita ao aluno responder questões, com base em análise dos códigos sobre assuntos como algoritmos e programação, desenvolvimento *web*, *Java*, entre outros. Este é um método interessante, pois juntamente com toda a teoria, o aluno testa os conhecimentos obtidos, respondendo a perguntas pontuais. Outra vantagem desta ferramenta é que ela é totalmente *online*, não dependendo do modelo do sistema operacional instalado no computador. A grande desvantagem da ferramenta é verificada em relação ao conteúdo escolhido, que se torna demasiadamente teórico, não possibilitando ao aluno colocar em prática seus conhecimentos adquiridos no momento. Por este motivo, se faz necessário o uso de ferramentas conhecidas como *IDEs* (*Integrated Development Environment*) ou Ambientes de Desenvolvimento de Programação.

Além destas ferramentas, algumas técnicas são utilizadas para aprimorar o processo de aprendizagem de programação e, entre elas, está a Gamificação, que será abordada a seguir.

2.6 Gamificação

Atrair a atenção dos alunos, utilizando o método tradicional de ensino, para que se interessem cada vez mais pelo conteúdo da disciplina de programação, sempre foi um grande desafio para os educadores. A partir desta dificuldade, surge o termo gamificação, ou em inglês, o *gamification.*

A gamificação consiste em fazer com que o aluno pense na mecânica do funcionamento de um jogo para conseguir resolver os problemas propostos. Como a gamificação é basicamente o funcionamento de um jogo, ela pode ser dividida em três partes: (i) as atividades a serem feitas para chegar aos objetivos, (ii) o progresso do jogador (ou aluno), (iii) e as recompensas a cada objetivo alcançado (KLOCK, 2014). Nesta mesma linha, Vianna (2013, p. 13) destaca que:

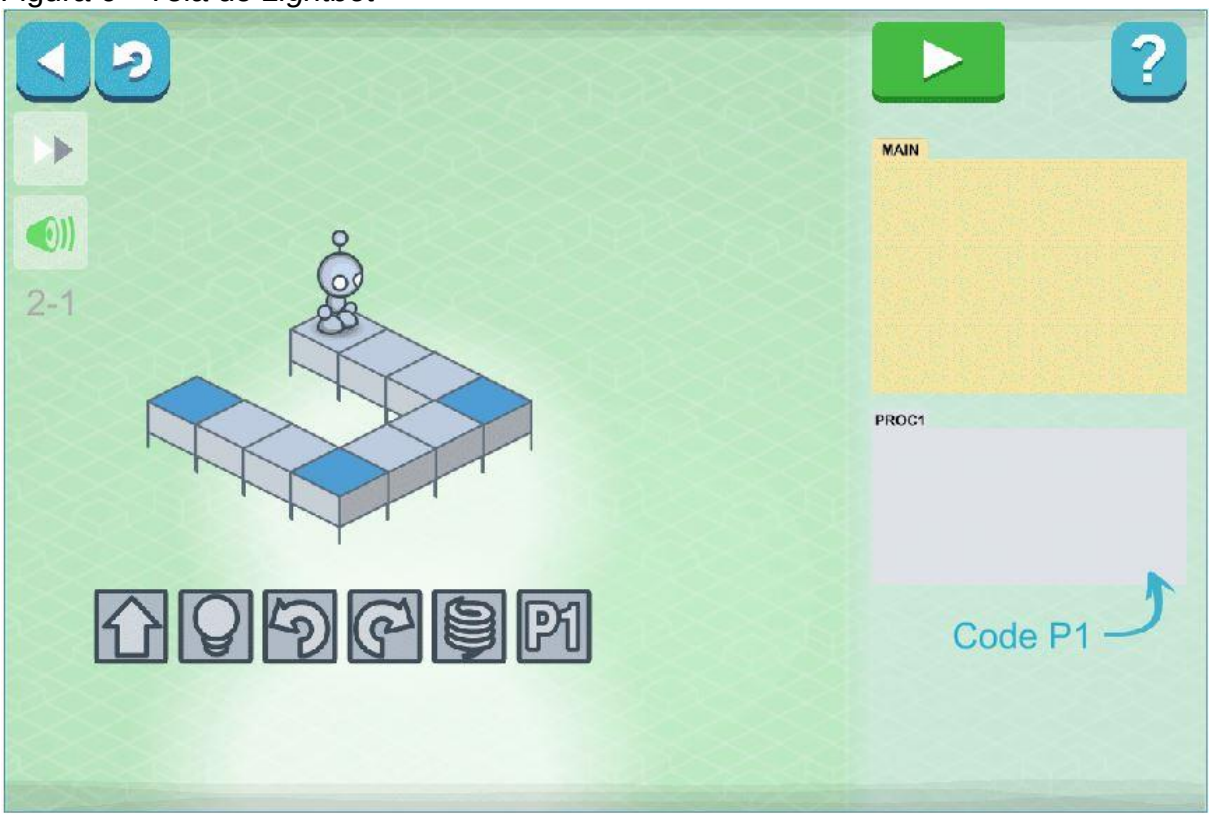
> A gamificação (do original em inglês gamification) corresponde ao uso de mecanismos de jogos orientados ao objetivo de resolver problemas práticos ou de despertar engajamento entre um público específico.

A gamificação utiliza algumas técnicas como: pontos, níveis, *rankings*, desafios e missões, medalhas ou conquistas, integração, *loops* de engajamento, personalização, reforço ou *feedback*, regras e narrativa (KLOCK, 2014).

Como exemplo, pode-se analisar algumas técnicas utilizadas por Lee Sheldon (2012), professor norte-americano e designer de *games*, que é autor do livro *Multiplayer Classroom: Designing Coursework as a Game. Ele s*aiu da indústria dos *games* e passou a ministrar uma disciplina chamada *game design,* que trata da introdução aos jogos eletrônicos. Ele passou a introduzir elementos de jogos ao andamento da disciplina, com o objetivo de engajar os alunos com o conteúdo a ser desenvolvido. Uma das técnicas utilizadas alterou o método de avaliação clássico, de modo que o aluno inicia a disciplina com a nota zero e a mesma é incrementada, de acordo com o andamento do semestre. As notas finais são contabilizadas através de um sistema de pontos, onde os mesmos são obtidos através de missões dadas aos alunos (SILVA, 2015).

Um exemplo de jogo que utiliza o conceito de gamificação é o *Lightbot* (LIGHTBOT, 2016). No jogo, o aluno controla um robô que deverá acender a lâmpada em todos os espaços azuis. Para concluir os desafios, são disponibilizadas algumas peças (comandos), que devem ser colocadas em sequência para que, quando o botão de "*play*" for clicado, o robô execute os passos corretos para atingir o objetivo. O jogo é dividido em módulos: básico, procedimentos e *loops* de repetição, que por sua vez são divididos em fases, como podem ser visto na Figura 6 (LIGHTBOT, 2016).

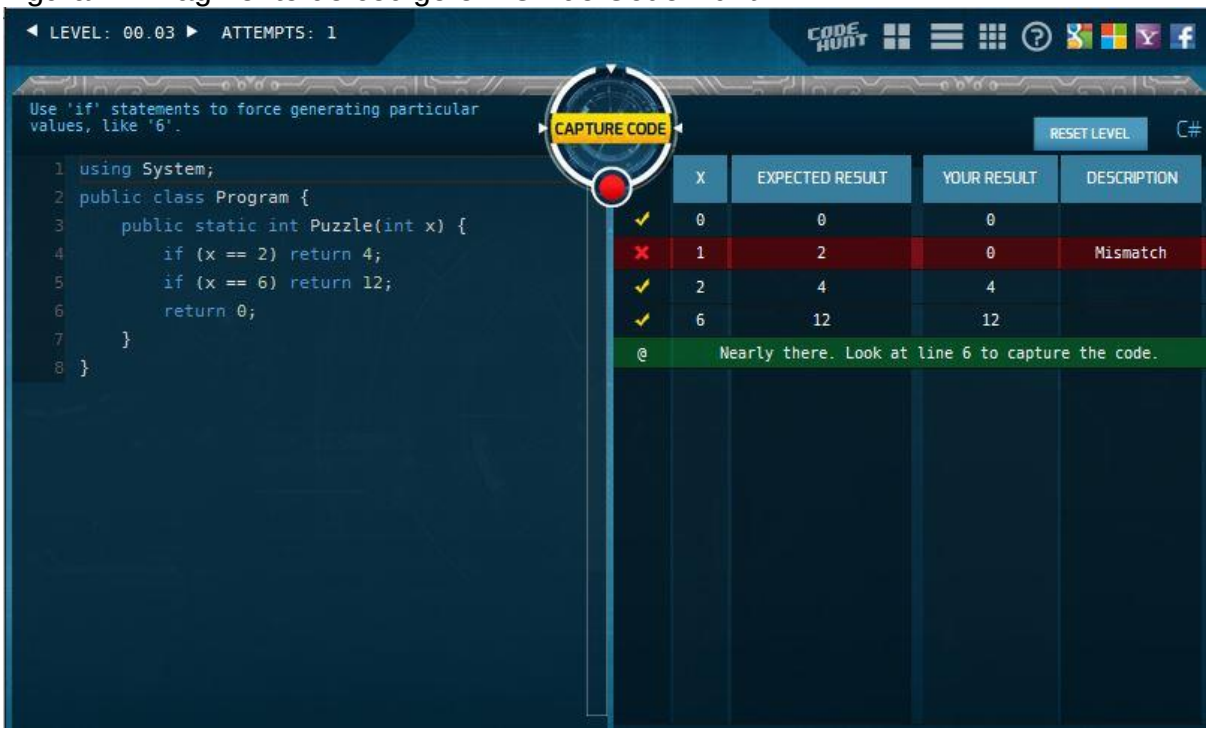
Figura 6 - Tela do *Lightbot*



Fonte: Lightbot, 2016.

Outro exemplo de jogo é o *Code Hunt* (CODE HUNT, 2016). Este jogo tem como objetivo consertar fragmentos de código de programação. Ao iniciar, o aluno deve escolher entre as duas opções de linguagem de programação: *Java* ou *C#.* O jogo é dividido em várias fases e em vários módulos que abordam diferentes conceitos de programação. Quando a fase é iniciada, é apresentado o fragmento de código que deve ser modificado para que volte a funcionar de acordo com as saídas exigidas pela fase, como pode ser observado na Figura 7 (CODE HUNT, 2016).

Figura 7 - Fragmento de código em *C#* do *Code Hunt*



Fonte: Code Hunt, 2016.

O uso da técnica da gamificação é muito útil, pois os desafios propostos pelos jogos, às vezes, são mais interessantes e compreensíveis do que os conteúdos da disciplina de algoritmos em si.

Além da técnica da gamificação, algumas outras técnicas foram implementadas com o objetivo de auxiliar não só os alunos, mas também os professores, como os juízes *online,* que serão detalhados na seção a seguir.

2.7 Juízes online

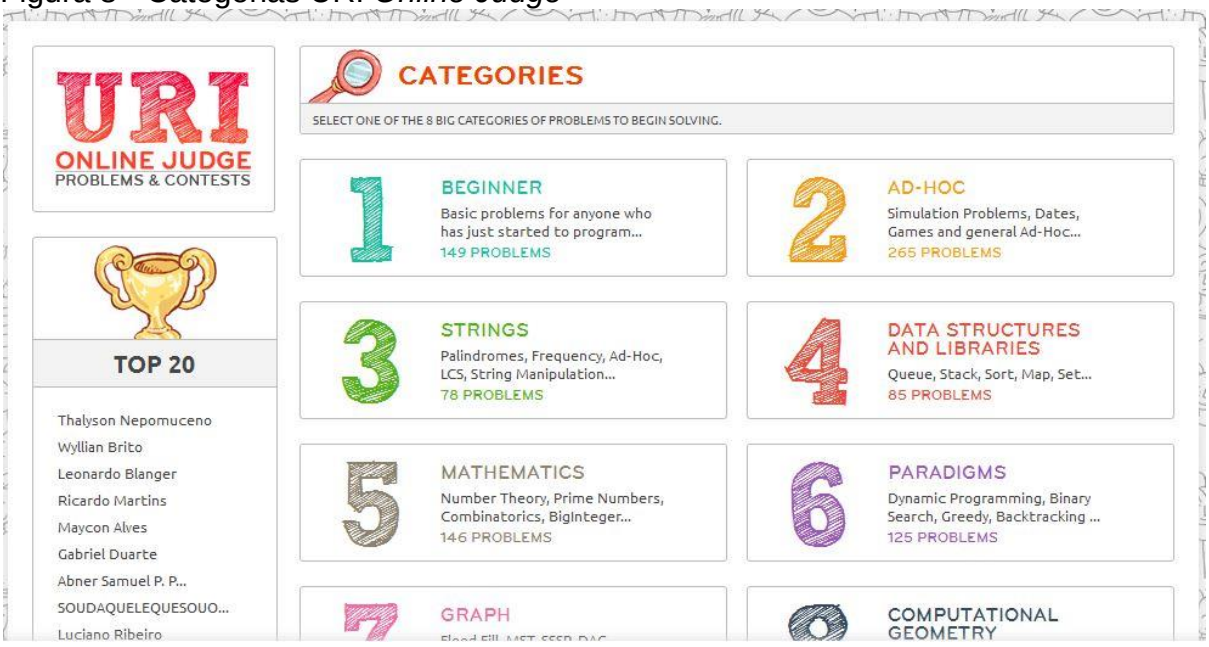
Os juízes *online* surgiram para suprir a demanda de correção e avaliação dos códigos-fonte escritos pelos alunos, utilizando alguma linguagem de programação. Esta ferramenta faz o papel do avaliador e, dentre outros, pode ter os seguintes resultados da análise: certo, errado, saída mal formatada, erro de compilação, erro em tempo de execução (CHAVES, et al., 2013).

Os programas, em sua grande maioria, precisam obter como entrada dados padronizados e formatados. A partir destes dados, o programa faz o processamento adequado. Concluída a fase do processamento, o programa está pronto para mostrar os resultados, e faz isso de uma forma padronizada e formatada.

Utilizando o princípio de que este processo é sempre o mesmo, é possível que a avaliação dos programas seja feita por uma ferramenta automatizada. A ferramenta de Juízes *Online* recebe o código-fonte enviado pelo aluno e faz a compilação deste código, seguido da execução do mesmo. Se nesta fase a ferramenta detectar algum erro, é retornado para o aluno o erro de compilação. Enquanto o programa está sendo executado, os Juízes colocam como entrada do programa os dados formatados e, após processá-los, são comparados com os resultados esperados (KURNIA, et al., 2001).

Dois exemplos de Juízes *Online* são: o SPOJ Brasil (SPOJ Brasil, 2016) e o URI *Online Judge* (URI Online Judge, 2016). As duas ferramentas trabalham de forma muito semelhante, tendo uma lista de problemas a serem solucionados, por onde os alunos submetem suas respostas, tendo o resultado de falha ou sucesso. Os problemas propostos são todos categorizados, cujo aluno pode escolher a de sua preferência, como pode ser visto na Figura 8.

Figura 8 - Categorias URI *Online Judge*

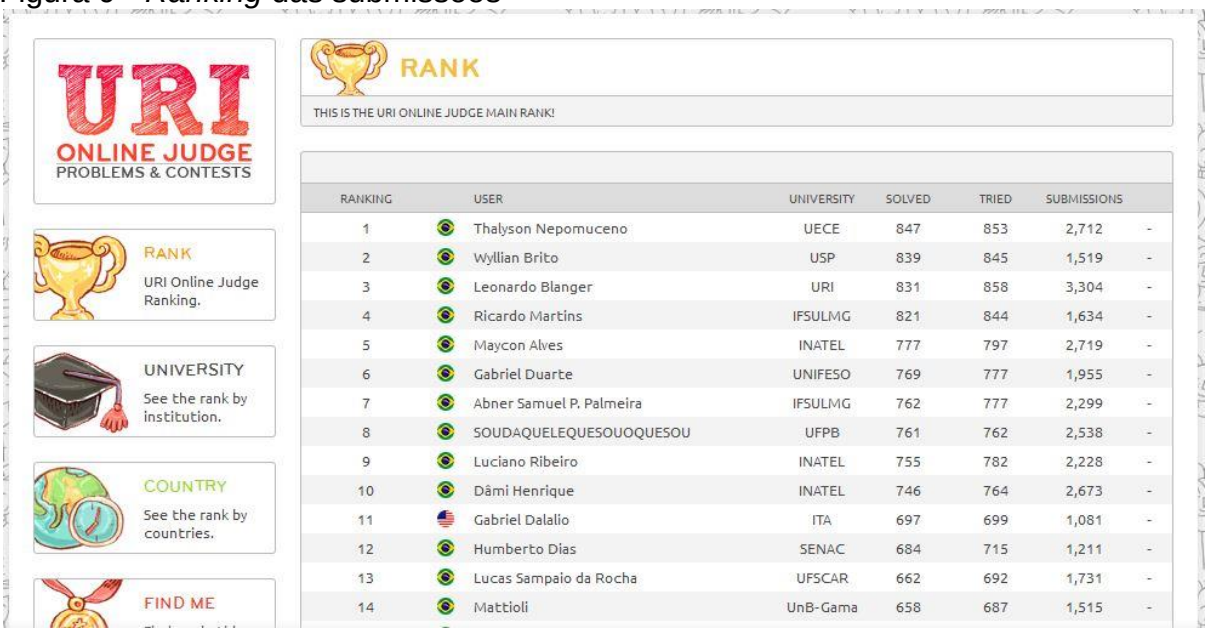


Fonte: URI Online Judge, 2016.

Os problemas propostos podem ser desde coisas do cotidiano, quanto às questões mais específicas de matemática, por exemplo. De acordo com as submissões enviadas, os alunos entram em um *ranking,* o qual é ordenado pela

quantidade de soluções corretas, como pode ser visto na Figura 9 (URI Online Judge, 2016).

Figura 9 - *Ranking* das submissões



Fonte: URI Online Judge, 2016.

Os Juízes *Online* facilitam muito o estudo extraclasse, substituindo o professor na avaliação do código fonte do programa escrito pelo aluno.

Dentre as ferramentas que focam no autoestudo, uma plataforma foi criada para suprir esta demanda: os *Massive Open Online Courses,* ou os *MOOCs.* Os *MOOCs* contam com vários cursos que são oferecidos a um grande número de pessoas, que podem participar de qualquer parte do mundo. A seção a seguir detalha o conceito de *MOOCs*.

2.8 Massive open online courses (MOOC)

Uma das plataformas de aprendizado que mais tem se desenvolvido nos últimos tempos é o *MOOC*. Isto se deve ao fato da popularização da internet e, consequentemente, o conhecimento ficou mais acessível às pessoas. Um *MOOC* pode ser considerado uma ferramenta *online* de autoestudo que auxilia o ensino de programação. O *MOOC* tem a premissa de disponibilizar acesso livre a cursos de qualidade que, por consequência, reduz os custos da educação e podem até modificar os modelos atuais de ensino (YUAN; POWELL, 2013).

Pelo motivo de serem *online*, os cursos podem ter um grande número de inscritos, pois não se limita a espaço físico de uma sala de aula. Um exemplo disto foi um curso disponibilizado pela Universidade de Stanford nos Estados Unidos, sobre Inteligência Artificial, que contou com mais de 160.000 inscritos. Na época, este fato teve grande repercussão no mundo todo. Esta iniciativa chamou a atenção dos funcionários e professores, que acabaram fundando a *Coursera*. A *Coursera* oferece cursos nas mais diversas áreas e trabalha em parceria com as universidades em diversos países, contando com mais de um milhão de alunos (ALBERTI, 2013).

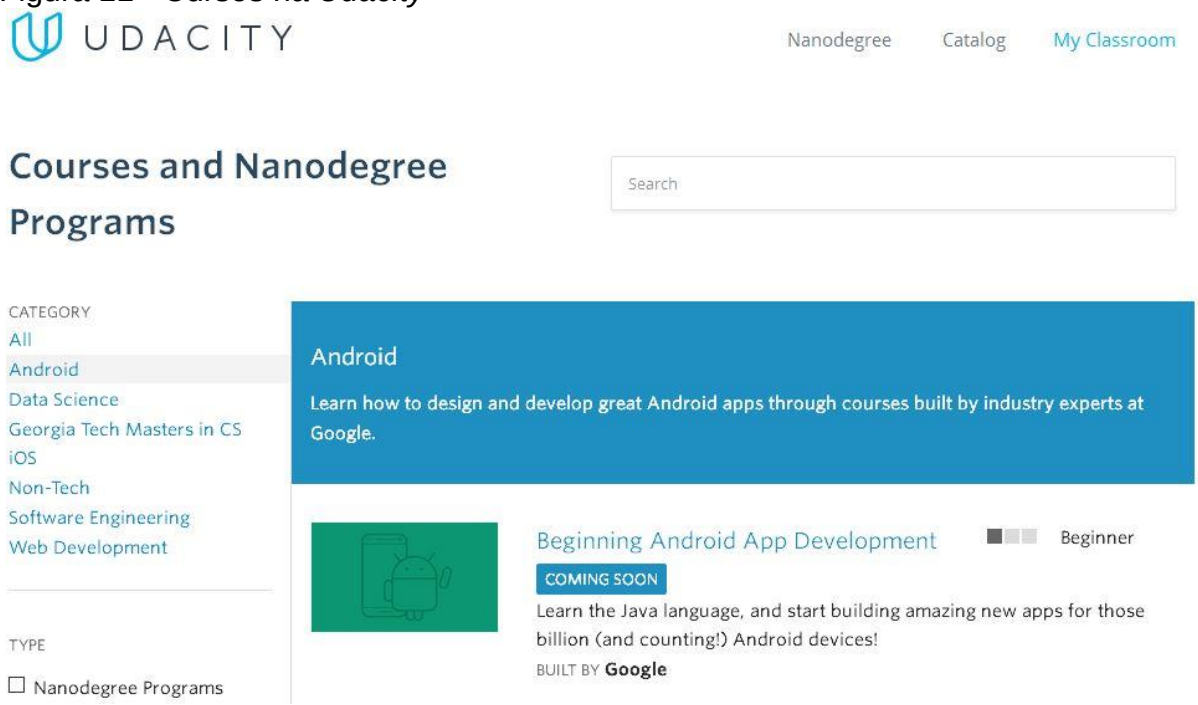
Exemplos de *MOOCs* podem ser vistos nas plataformas *Coursera* (COURSERA, 2016) e *Udacity* (UDACITY, 2016), onde podem ser encontrados cursos das mais diversas áreas. Alguns exemplos de cursos são de desenvolvimento para a plataforma *Android*, desenvolvimento *web* e Engenharia de *Software* que são oferecidos na *Udacity*. Cursos como Introdução à Lógica, Programação em *Java* e Introdução ao Controle de Sistemas, que são utilizados nos cursos de Engenharia, podem ser encontrados na plataforma *Coursera*. As plataformas de *MOOCs* possuem cursos gratuitos ou pagos, cujo objetivo é complementar o estudo presencial.

A *Coursera* e a *Udacity* foram criadas com o objetivo de prover educação de alto nível, disponibilizando cursos totalmente *online*, ao maior número de pessoas possível. Para isso, foram feitas parcerias com várias universidades e instituições de ensino de todo o mundo. Um dos diferenciais deste método de ensino é que cada um aprende no seu ritmo, assistindo as videoaulas e fazendo as tarefas propostas. A *Udacity* trabalha com a premissa de que a educação é um direito básico humano, para fazer com que os alunos progridam em suas carreiras, capacitando-os. A plataforma *Coursera* trabalha com quatro fundamentos básicos, que são: a eficácia do aprendizado online, pedagogia de domínio, avaliação entre colegas e ensino híbrido. De acordo com os interesses selecionados pelo aluno, os cursos disponíveis são listados, como pode se observar nas figuras 10 e 11 (COURSERA, 2016) (UDACITY, 2016).

Figura 10 - Cursos disponíveis na *Coursera*

Fonte: Coursera, 2016.

Figura 11 - Cursos na *Udacity*



Fonte: Udacity, 2016.

Com o intuito de agregar ainda mais conhecimento e estimulando o estudo extraclasse, entram em cena os geradores de exercícios de programação. A maioria dos geradores trabalha com uma lista estática de exercícios. Com isso, há uma demanda por geradores automáticos, conceito abordado na seção a seguir.

2.9 Geradores de exercícios de programação

Na atualidade, existem várias ferramentas de auxílio ao ensino de programação. Algumas delas como o *VisuAlg* (*VISUALG*, 2016), *Robocode* (*ROBOCODE*, 2016), *Codecademy* (*CODECADEMY*, 2016) são muito utilizadas e trabalham basicamente da mesma forma, com listas estáticas de exercícios.

Todas estas ferramentas e plataformas são de grande utilidade, mas todas elas têm suas deficiências. Os *MOOCs* por exemplo, têm como ponto positivo, cursos que antes só eram disponibilizados em sala de aula. Agora podem ser vistos de qualquer parte do mundo e por um grande número de pessoas. Como ponto negativo, temos que o conteúdo tem poucas práticas e os alunos têm de seguir exemplos, não tendo uma orientação precisa sobre a resolução dos exercícios. Os Juízes *online* servem como analisadores de código e não como geradores de exercícios, exigindo que o aluno escreva um programa para os Juízes analisarem. Pensando nestas deficiências, identificou-se a necessidade de uma ferramenta mais completa, onde o aluno tem acesso a exercícios gerados para resolver, escreve a solução e pode analisar de forma mais completa o código.

Além de estas ferramentas servirem de apoio à aprendizagem, algumas técnicas estão sendo difundidas e aperfeiçoadas pelo mundo acadêmico, como por exemplo, a gamificação, visando aumentar a compreensão do conteúdo das disciplinas de programação. Após algumas pesquisas, foi observado que, de fato, há uma carência por ferramentas que gerem exercícios de programação automaticamente, fazendo com que aumente muito a possibilidade de diferentes enunciados de exercícios (TRAMONTINA, 2015).

Ciente desta carência, Tramontina (2015) desenvolveu um gerador automático de exercícios de programação. Este trabalho pretende estender as funcionalidades do sistema gerador original, no qual será implementado o conceito de vetores, matrizes, procedimentos e funções, para que os alunos das disciplinas de algoritmos e programação tenham uma melhor experiência e possam praticar todos os conteúdos abordados na disciplina.

No Capítulo a seguir, a metodologia utilizada na implementação dos recursos estendidos do Gerador Automático de Exercícios de Programação é apresentada em detalhes.

3 METODOLOGIA

Conforme o pensamento de Knechtel (2014), a metodologia científica é a observação dos fenômenos da nossa realidade, de forma sistemática, obedecendo a um rígido controle das informações, através de uma sequência de passos orientados pela teoria, que por sua vez, visa explicar todos estes fenômenos.

Para o desenvolvimento do presente trabalho, será feita uma pesquisa qualitativa, definida como a ação de fazer questionamentos e organizar o conhecimento adquirido, como observa Knechtel (2014, p. 81):

> É o ato de pesquisar, interrogar, questionar e sistematizar o conhecimento. Como a pesquisa pressupõe obter fatos e fenômenos na realidade, fica claro que partimos de fatos menos elaborados e chegamos a um fato ou efeito mais elaborado.

A pesquisa qualitativa, utilizada no desenvolvimento dos recursos estendidos do gerador automático de exercícios, é um tipo de pesquisa que visa à compreensão de fenômenos e seu intuito é ter uma percepção mais detalhada e complexa dos mesmos, através das respostas dos informantes aos questionamentos feitos. Neste tipo de pesquisa, o papel dos informantes é de grande relevância (KNECHTEL, 2014).

3.1 Recursos originais do gerador automático de exercícios para apoio ao ensino de programação

O sistema gerador, desenvolvido por Tramontina (2015), conta com três componentes principais: o gerador, o tradutor e a lista com os *templates.* Com estes componentes o sistema pode gerar uma lista de exercícios, cuja lista é

composta pelo título, enunciado e a solução, que pode ser na linguagem *Java*, *C*, *C++* e *Phyton*.

Os *templates* são arquivos estruturados utilizados como base para a geração dos exercícios. Nestes está detalhado o nível de dificuldade do exercício, o enunciado genérico e a solução formatada em pseudocódigo. No corpo do *template,* há lacunas que são preenchidas aleatoriamente no procedimento de geração do exercício, logo, cada *template* equivale a um exercício. Estas lacunas permitem que seja gerada uma quantidade diversificada de exercícios (TRAMONTINA, 2015).

Cada exercício possui algumas informações básicas como o código do *template*, o nível de dificuldade, o tópico principal e os rótulos do exercício. A função do código é tornar cada exercício único. O nível de dificuldade identifica a complexidade do exercício e o tópico indica a qual assunto o exercício se refere. Para que o tradutor possa interpretar o *template* corretamente, o mesmo deve possuir estas informações básicas, escritas em inglês (TRAMONTINA, 2015).

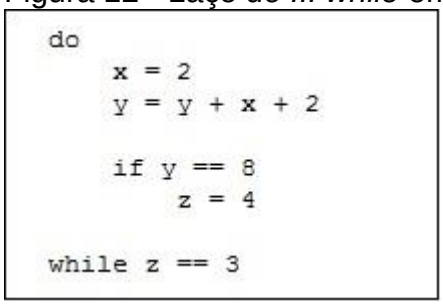
3.2 Recursos estendidos do gerador automático de exercícios para apoio ao ensino de programação

Este trabalho pretende estender os seguintes recursos do sistema gerador: aprimoramento do pseudocódigo para suportar exercícios com laços de repetição *do ... while,* aplicação do conceito de vetores e matrizes; implementação de recursos para possibilitar o uso de procedimentos e funções.

As implementações a serem feitas terão impacto direto nos arquivos de *template*. O pseudocódigo no qual o exercício é escrito, incorporará alguns comandos. A sintaxe dos comandos adicionais segue o mesmo padrão dos comandos já existentes.

Para criar um bloco de laço de repetição, utilizando o comando *do ... while*, a seguinte sintaxe do pseudocódigo deverá ser seguida: *do* <nova linha> <endentação de 4 espaços> <conjunto de instruções> <nova linha> <recuo de 4 espaços> *while* <condição>. A sintaxe do comando é muito parecida com a utilizada nas linguagens *Java*, *C* e *C++*, mas conta com alguns detalhes, como por exemplo, não faz uso de parênteses e ponto e vírgula. Na Figura 12, pode ser observado como fica o laço *do ... while* em pseudocódigo.

Figura 12 - Laço *do ... while* em pseudocódigo



```
do
    x = 2
    y = y + x + 2

    if y == 8
        z = 4

while z == 3
```

Fonte: elaborado pelo autor.

A declaração dos vetores em pseudocódigo deve ser feita, utilizando a seguinte sintaxe: *declare array* <variável> *of* <tipo> *size* <tamanho>, onde <variável> deve ser o nome do vetor, <tipo> deve ser os tipos primitivos de variáveis e <tamanho> que será o tamanho do vetor. O uso dos vetores em pseudocódigo é muito semelhante aos usos vistos nas linguagens de programação comerciais. Na Figura 13, pode ser observado um exemplo de exercício em pseudocódigo que faz uso de vetores.

Figura 13 - Exercício com vetor em pseudocódigo

```
declare array iArray of integer size 10
declare integer i

while i < 10
    iArray[i] = random(1,10)
    i = i + 1

i = 0

while i < 10
    print iArray[i]
    i = i + 1
```

Fonte: elaborado pelo autor.

As matrizes em pseudocódigo são muito semelhantes aos vetores, contudo, são bidimensionais, com a identificação de índices para linhas e colunas. A declaração de uma matriz é feita usando a sintaxe *declare matrix* <variável> *of* <tipo> *size* <linhas> *by* <colunas>, onde o que muda da declaração dos vetores é a <linha> que se refere ao número de linhas e a <coluna> que reflete a quantidade de

colunas que a matriz terá. Na Figura 14, está detalhado um exemplo de exercício, semelhante ao exercício da Figura 13, porém, utilizando o conceito de matrizes.

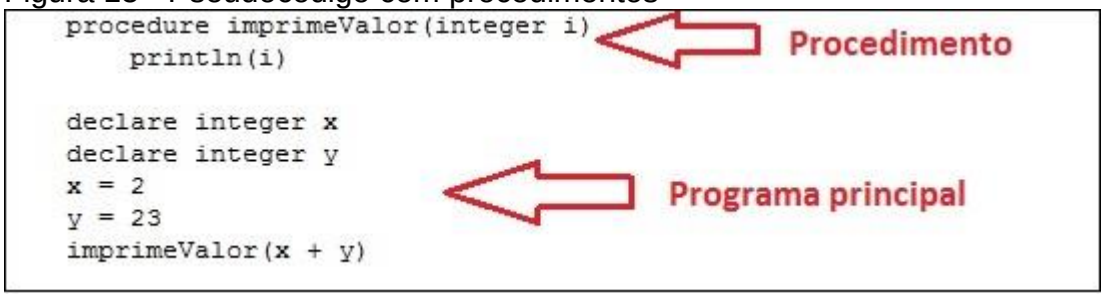
Figura 14 - Exercício com matriz em pseudocódigo

```
declare matrix iMatrix of integer size 10 by 10
declare integer i
declare integer j

while i < 10
    while j < 10
        iMatrix[i][j] = random(1,10)
        j = j + 1
    i = i + 1
    j = 0

i = 0

while i < 10
    while j < 10
        print iMatrix[i][j]
        j = j + 1
    i = i + 1
    j = 0
    println ""
```

Fonte: elaborado pelo autor.

Os procedimentos são trechos de código, que podem receber parâmetros, mas não possuem nenhum retorno. Nos *templates* do sistema gerador o uso de procedimentos será possível, utilizando a sintaxe *procedure* <nome do procedimento> (<parâmetros>), onde <nome do procedimento> deverá ser o nome a ser chamado no trecho principal do programa e <parâmetros> deverá ser a lista de variáveis que o procedimento irá receber. Esta lista deve seguir a estrutura de tipo da variável e nome da variável e podem conter mais de uma variável, mas sempre separadas por vírgulas. Após a declaração, segue o trecho de código do procedimento, que deverá seguir a mesma lógica de endentação do trecho principal do programa. Como uma boa prática, os procedimentos deverão sempre vir antes do trecho do programa principal, como pode ser observado na Figura 15.

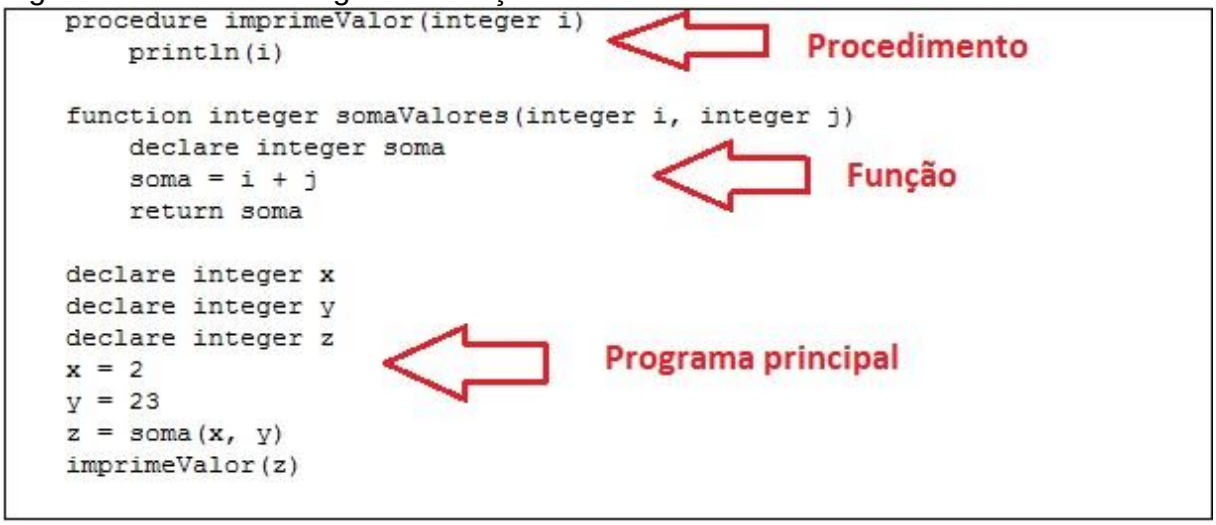
Figura 15 - Pseudocódigo com procedimentos



Fonte: elaborado pelo autor.

As funções são muito semelhantes aos procedimentos, com a diferença de que as funções retornarão algum valor ao fim de sua execução. Sua sintaxe em pseudocódigo é também muito semelhante ao dos procedimentos. Deverá seguir a seguinte sintaxe: *function* <tipo do retorno> <nome da função> (<parâmetros>), onde <tipo do retorno> deverá ser o tipo (*integer*, *real*, *string*...) do dado de retorno, seguido do <nome da função>, que será o nome a ser chamado no trecho do programa principal e os parâmetros que segue a mesma lógica dos parâmetros dos procedimentos, citados anteriormente. A estrutura e sintaxe de uma função podem ser observadas na Figura 16.

Figura 16 - Pseudocódigo com funções



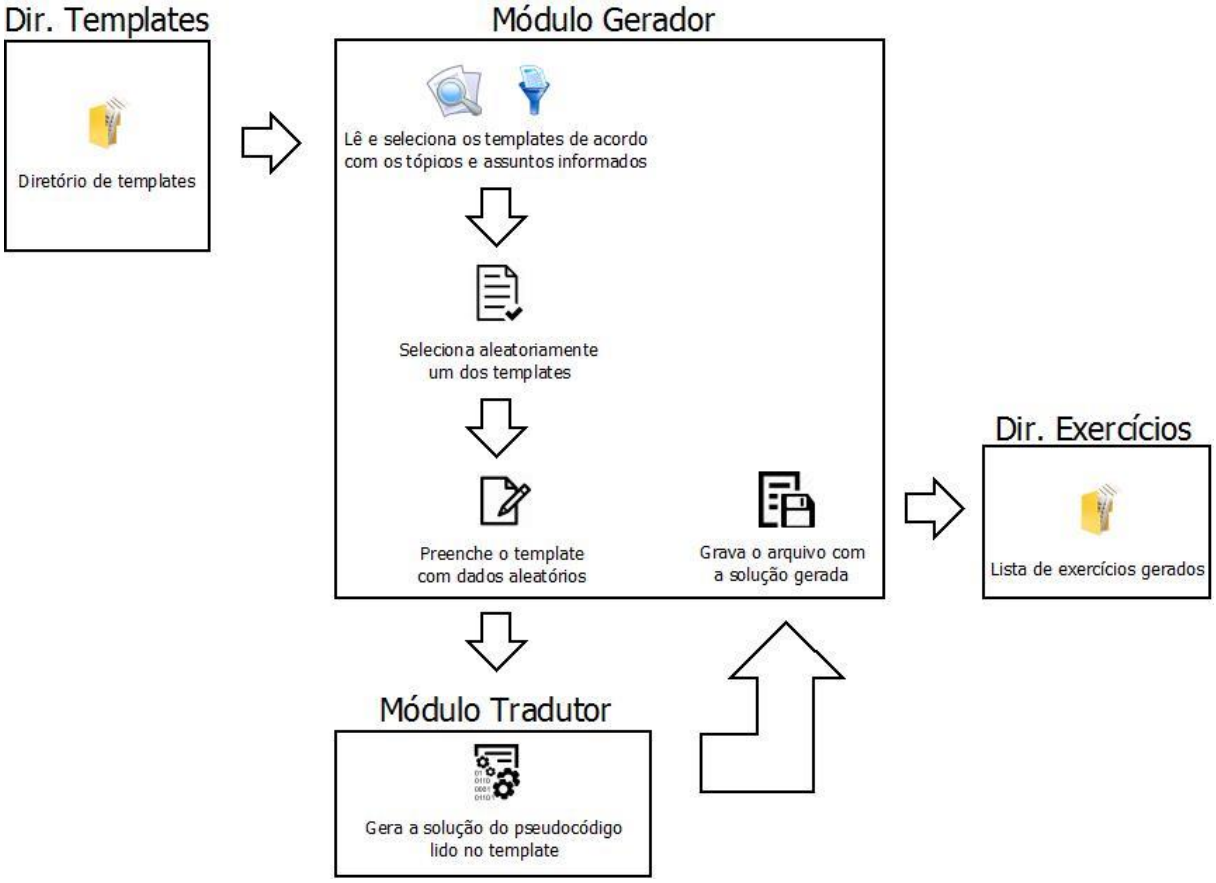
Fonte: elaborado pelo autor.

Após a apresentação dos novos comandos, no Capítulo seguinte, é apresentada uma visão geral do sistema gerador. Neste Capítulo, podem ser observadas as partes do sistema que serão afetadas pelas novas implementações e também comparações de sintaxe do pseudocódigo antes e depois das implementações.

4 VISÃO GERAL DO GERADOR AUTOMÁTICO DE EXECÍCIOS

O sistema gerador conta com três componentes principais: o gerador, o tradutor e a lista de *templates*. Cada um destes componentes tem papel fundamental na geração dos exercícios.

Figura 17 - Processo de geração dos exercícios



Fonte: elaborado pelo autor.

A Figura 17, detalha o processo pelo qual os exercícios são gerados e, como pode ser observado, o funcionamento do sistema gerador inicia com uma lista de

templates. Cada *template* irá gerar um exercício (TRAMONTINA, 2015). O processo pelo qual os exercícios serão gerados, não terá a necessidade de ser alterado, pelo fato de que os novos recursos e funcionalidades serão implementados no módulo tradutor e, desta forma, o processo seguirá a estratégia traçada por Tramontina (2015).

O módulo gerador tem a função de ler a lista de *templates* e selecioná-lo, de acordo com os parâmetros informados pelo usuário. Depois de selecionar um exercício, o módulo gerador faz a leitura do conteúdo do *template*, preenchendo-o com dados aleatórios e, após esta etapa, o conteúdo preenchido é enviado ao módulo tradutor (TRAMONTINA, 2015).

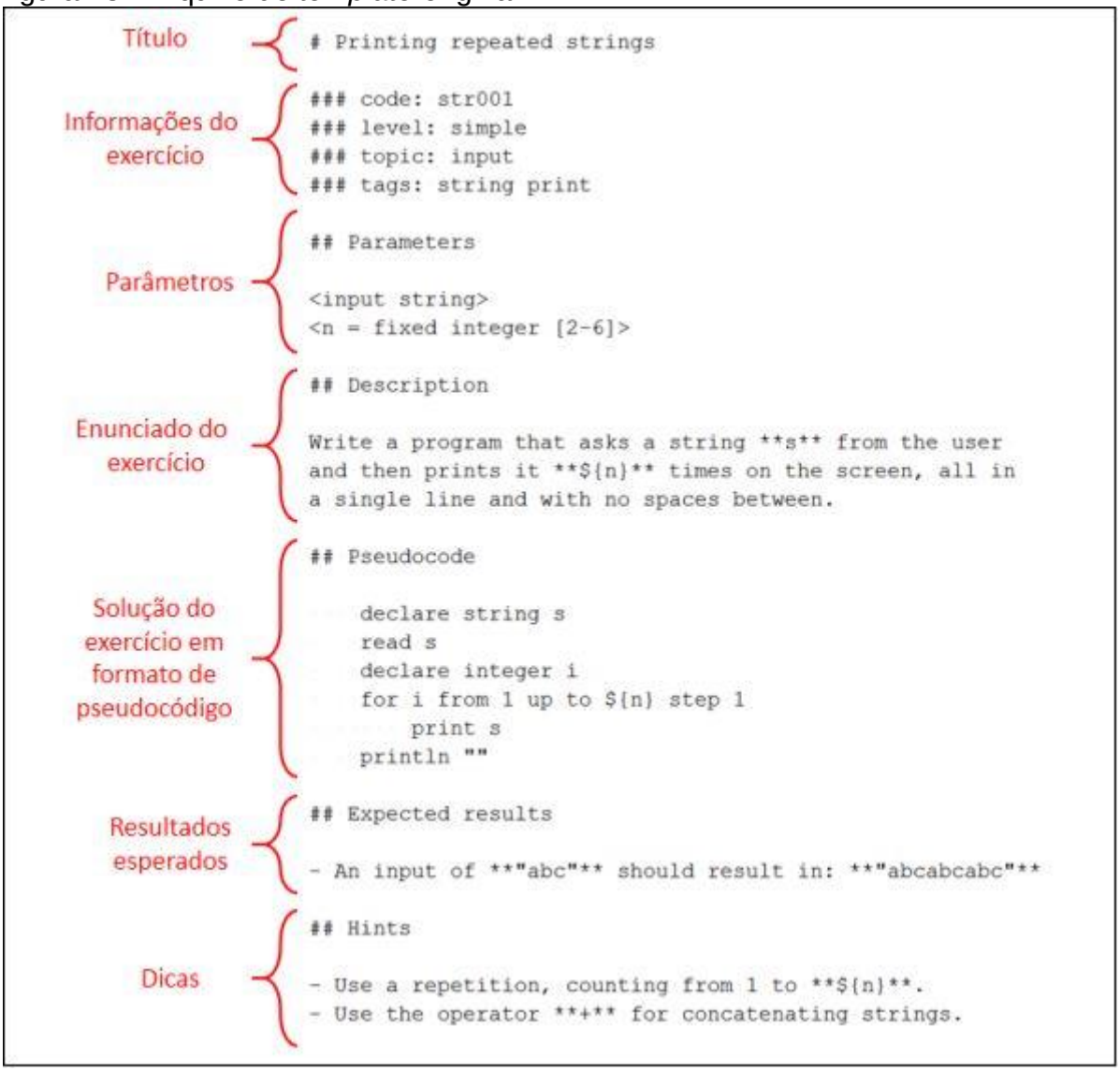
O módulo tradutor tem como função principal traduzir o pseudocódigo recebido pelo módulo gerador, alterando os comandos no formato de pseudocódigo para o formato da linguagem de programação definida pelo usuário, gerando a solução do exercício. Superada esta etapa, o exercício já formatado na linguagem parametrizada, volta para o módulo gerador, que por sua vez, faz a gravação do arquivo. Este processo se repete para cada exercício gerado. A solução do exercício é gerada pelo módulo tradutor, a partir do trecho de pseudocódigo, informado no *template*, conforme observa Tramontina (2015, p. 39):

> Para gerar a solução, o módulo tradutor faz uso do pseudocódigo contido dentro do *template*. Este pseudocódigo é a solução do exercício, e tem sua estrutura semelhante ao Portugol, porém, com termos criados na língua inglesa. A proposta de usar um pseudocódigo para gerar a solução do exercício foi utilizada com o intuito de auxiliar no ensino de programação.

Como a solução do exercício é baseada no pseudocódigo, este trecho pode ser disponibilizado ao aluno, caso apresentar dificuldades na resolução do exercício. Outra vantagem do uso do pseudocódigo é que o módulo tradutor pode gerar o exercício em várias linguagens de programação, pois a solução em pseudocódigo é uma solução genérica.

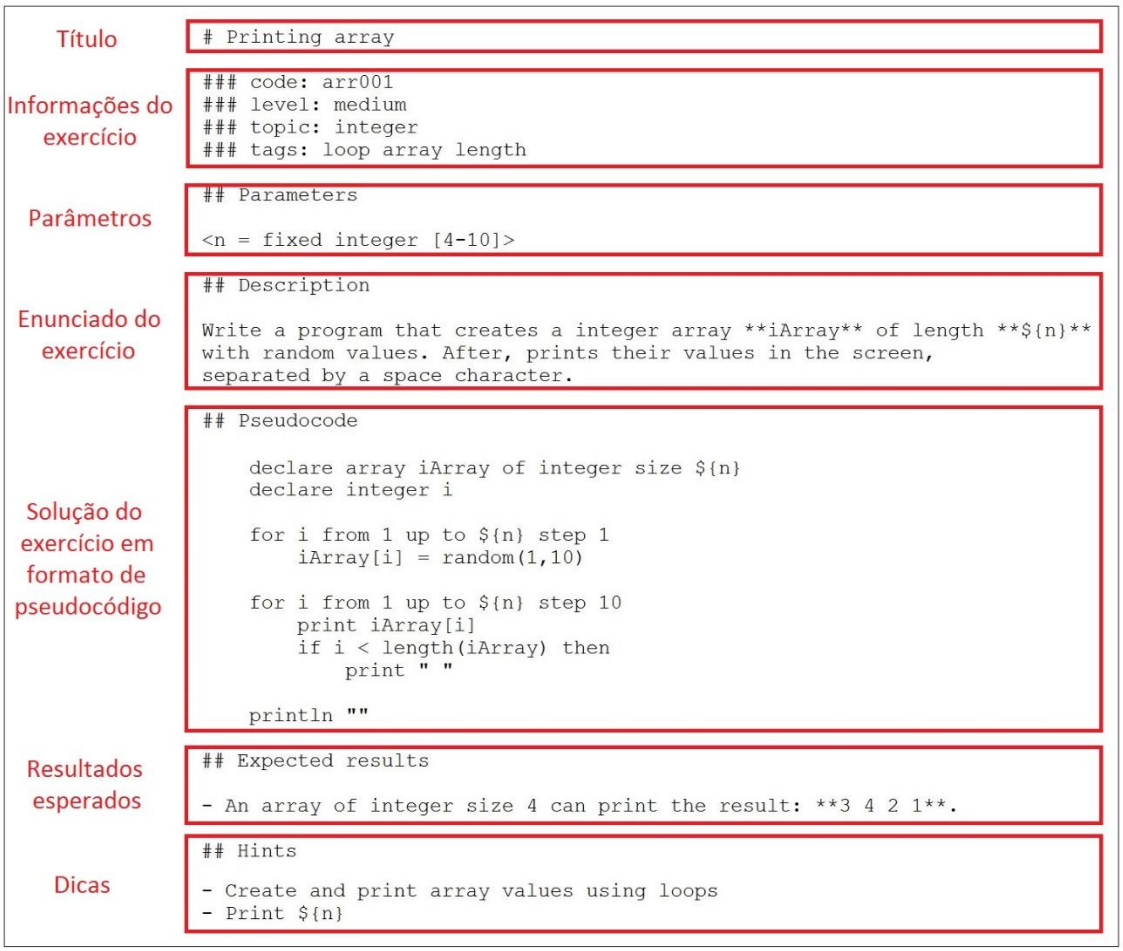
Os *templates* armazenam as informações que os exercícios necessitam para serem gerados. São arquivos-texto que devem ser criados manualmente e armazenados no diretório específico dos *templates*. A estrutura básica de um arquivo de *template* pode ser observada nas Figuras 18 e 19.

Figura 18 - Arquivo de *template* original



Fonte: Tramontina, 2015.

Figura 19 - Protótipo do novo arquivo de *template*



Fonte: elaborado pelo autor.

Na Figura 18, observa-se a estrutura de um *template* antes das implementações propostas pelo presente trabalho. Na Figura 19, está sendo mostrado o *template* de um exercício que envolve laços de repetição e vetores. É declarado um parâmetro "***n***" que é utilizado para o preenchimento do exercício. O aluno deverá criar um programa que cria um vetor "***iArray***" de números inteiros, e o preenche com valores aleatórios. Depois de preenchido, o aluno deve imprimi-lo na tela, separando os valores por um espaço.

O arquivo de *template* deve ser criado, seguindo uma estrutura específica para garantir seu correto funcionamento, como observa Tramontina (2015, p. 40):

> Sua estrutura segue o formato de uma linguagem de marcação, mais especificamente, usando a sintaxe Markdown (DARING FIREBALL, 2016). A proposta de utilizar uma linguagem como esta ocorreu em virtude da saída ser gerada pronta para uso. Desta forma, o arquivo já é criado com uma formatação básica, que provém da própria linguagem de

> marcação, bastando apenas disponibilizar o exercício criado em um ambiente que formata o texto com base nesta sintaxe.

A sintaxe da *Markdown*[1] possui alguns caracteres especiais que servem para delimitar as áreas do texto, como por exemplo, o caractere “#”, que é utilizado para indicar o título do nível. O *template* deve seguir uma estrutura predefinida e utilizar a sintaxe da *Markdown* para que o módulo tradutor entenda os comandos e traduza na linguagem definida pelo usuário (TRAMONTINA, 2015).

As informações que compõem o *template* começam pelo título, que é precedido do caractere “#” e um espaço. A seguir, temos as informações do exercício, que são precedidas pelos caracteres “###”. Neste ponto temos o código do *template* precedido pela palavra *code*, o nível de dificuldade precedido pela palavra *level*, o assunto principal do exercício precedido pela palavra *topic* e os assuntos relacionados a ele, precedidos pela palavra *tags* (TRAMONTINA, 2015).

O *template* conta com uma seção disponível para os parâmetros informados, que servem como base da geração dos exercícios, como pode ser observado na Figura 20.

Figura 20 - Parâmetros de geração dos exercícios

Parâmetro	Descrição do parâmetro
<n = fixed integer [2 – 6]>	Variável “n” recebe um inteiro entre 2 e 6.
<c = fixed char [a – z]>	Variável “c” recebe um caractere entre ‘a’ e ‘z’.
<d = fixed double [0.0 – 2.0]>	Variável “d” recebe um número real entre 0.0 e 2.0.
<s = fixed string [“abc”, “def”, “ghi”]>	Variável “s” recebe uma das strings dentro dos colchetes.
<b = fixed boolean >	Variável “b” recebe verdadeiro ou falso.

Fonte: Tramontina, 2015.

Os parâmetros têm como característica o caractere “<”, que significa o início de um parâmetro e, para finalizar, utiliza-se o caractere “>”. Podem ser utilizados parâmetros de entrada e de preenchimento, mas o sistema gerador utiliza somente os parâmetros de preenchimento para a geração (TRAMONTINA, 2015). Nos parâmetros, não há necessidade de alterações para adicionar os novos recursos,

[1] Markdown é uma ferramenta de conversão de texto em HTML ou XHTML. Ela permite escrever, utilizando um formato de texto simples de fácil leitura e escrita (DARING FIREBALL, 2016).

pois os mesmos são utilizados somente para preenchimento dos enunciados e lacunas no pseudocódigo dos exercícios.

A próxima seção do *template* está destinada à descrição do enunciado do exercício. No enunciado, os parâmetros informados na seção anterior podem ser utilizados em qualquer momento, sendo precedidos da sintaxe “${“, nome do parâmetro e finalizando com o caractere “}” (TRAMONTINA, 2015).

Após o enunciado do exercício, o *template* conta com a solução do mesmo em formato de pseudocódigo. Sua estrutura e sintaxe foram criadas por Tramontina (2015), e se assemelham às estruturas das linguagens de programação. Na Figura 21, pode-se ver o comparativo do pseudocódigo e da linguagem *Java*.

Figura 21 - Comparação do pseudocódigo com a linguagem *Java*

PSEUDOCÓDIGO	JAVA
TIPOS	
boolean	boolean
character	char
integer	int
real	double
string	String
OPERADORES	
+ - * / %	+ - * / %
== != < <= > >=	== != < <= > >=
not and or	! && ||
CONSTANTES	
true	true
false	false
pi	Math.PI
FUNÇÕES PREDEFINIDAS	
power(<a>, <b>)	Math.pow(a, b)
random(<a>, <b>)	(int)((-a + b + 1) * Math.random() + a)
COMANDOS	
declare <type> <var>	<type> <var>;
<var> = <value>	<var> = <value>;
read <var>	<var> = s.realLine();
print <var> , <value>	System.out.print (<var> + <value);
println <var> , <value>	System.out.println (<var> + <value);
if <cond> then ... else ...	if (<cond>) { ... } else { ... }
while <cond> ...	while (<cond>) { ... }
for <var> from <a> up to <b> step <c>	for (<var>=<a>; <var><=<b>; <var>=<var>+<c>) { ... }
for <var> from <a> down to <b> step <c>	for (<var>=<a>; <var><=<b>; <var>=<var>+<c>) { ... }

Fonte: Tramontina, 2015.

Para o desenvolvimento dos recursos estendidos, algumas modificações no pseudocódigo se fizeram necessárias. Nas Figuras 22 e 23, podem ser observados todos os comandos, com destaque para os comandos estendidos. Os recursos já implementados e os a serem estendidos, com suas respectivas traduções para as linguagens *Java* e *C,* estão detalhados na Figura 22 e, para *C++* e *Python,* na Figura 23.

Figura 22 - Comparação do pseudocódigo em *Java* e *C*

PSEUDOCODE	JAVA	C
TIPOS		
boolean	boolean	int
character	char	char
integer	int	int
real	double	double
string	String	char*
<var> as array of <n> <type>	<type>[] <var> = new <type>[n]	<type> <var>[n]
<var> as matrix of <n> by <m> <type>	<type>[][] <var> = new <type>[n][m]	<type> <var>[n][m]
OPERADORES		
+ - * / %	+ - * / %	+ - * / %
== != < <= > >=	== != < <= > >=	== != < <= > >=
not and or	! && ||	! && ||
CONSTANTES		
true	true	TRUE
false	false	FALSE
pi	Math.PI	M_PI
FUNÇÕES PREDEFINIDAS		
power(<a>, <b>)	Math.pow(a, b)	pow(a, b)
random(<a>, <b>)	(int)((-a + b + 1) * Math.random() + a)	rand() % (-a + b + 1) + a
length(<var>)	<var>.length()	strlen(<var>)
substring(<var>, <a>, <b>)	<var>.substring(<a>, <b>)	strncpy(<var>, <a>, <b>)
sqrt(<a>)	Math.sqrt(<a>)	sqrt(<a>)
COMANDOS		
declare <type> <var>	<type> <var>	<type> <var>
<var> = <value>	<var> = <value>	<var> = <value>
read <var>	<var>.readLine()	scanf(<var>, <a>)
print <var> , <value>	System.out.print(<var> + <value>)	printf(<var> + <value>)
println <var> , <value>	System.out.println(<var> + <value>)	printf(<var> + <value> + "\n")
if <cond> then ... else ...	if (<cond>) { ... } else { ... }	if (<cond>) { ... } else { ... }
while <cond> ...	while (<cond>) { ... }	while (<cond>) { ... }
for <var> from <a> up to <b> step <c>	for (<var>=<a>; <var><=<b>; <var>+=<c>) { ...	for (<var>=<a>; <var><=<b>; <var>+=<c>) { ...
for <var> from <a> down to <b> step <c>	for (<var>=<a>; <var><=<b>; <var>-=<c>) { ... }	for (<var>=<a>; <var><=<b>; <var>-=<c>) { ... }
do ... while <cond>	do { ... } while (<cond>)	do { ... } while (<cond>)
return <value>	return <value>	return <value>
exit	System.exit(0)	exit(0)
PROCEDIMENTOS, E FUNÇÕES		
procedure <name> (<parameters list>)	public void <name> (<parameters list>)	void <name> (<parameters list>)
function <type> <name> (<parameters list>)	public <type> <name> (<parameters list>)	<type> <name> (<parameters list>)

Fonte: elaborado pelo autor.

Figura 23 - Comparação do pseudocódigo em *C++* e *Python*

PSEUDOCODE	C++	PYTHON
TIPOS		
boolean	bool	-
character	char	-
integer	int	-
real	double	-
string	string	-
<var> as array of <n> <type>	<type> <var>[n]	[0 for i in range(n)]
<var> as matrix of <n> by <m> <type>	<type> <var>[n][m]	[[0 for i in range(n)] for i in range(m)]
OPERADORES		
+ - * / %	+ - * / %	+ - * / %
== != < <= > >=	== != < <= > >=	== != < <= > >=
not and or	! && ||	not and or
CONSTANTES		
true	TRUE	True
false	FALSE	False
pi	M_PI	math.pi
FUNÇÕES PREDEFINIDAS		
power(<a>, <b>)	pow(a, b)	a**b
random(<a>, <b>)	rand() % (-a + b + 1) + a	random.randint(a, b)
length(<var>)	<var>.length()	len(<var>)
substring(<var>, <a>, <b>)	<var>.substr(<a>, <b>)	<var>[<a>:<a>+<b>]
sqrt(<a>)	sqrt(<var>)	math.sqrt(<var>)
COMANDOS		
declare <type> <var>	<type> <var>	-
<var> = <value>	<var> = <value>	<var> = <value>
read <var>	getline(<var>, <a>)	<var> = raw_input()
print <var> , <value>	printf(<var> + <value>)	print <var>
println <var> , <value>	printf(<var> + <value> + "\n")	print <var> + '\n'
if <cond> then ... else ...	if (<cond>) { ... } else { ... }	if <cond>: ... else: ...
while <cond> ...	while (<cond>) { ... }	while <cond>: ...
for <var> from <a> up to <b> step <c>	for (<var>=<a>; <var><=<b>; <var>+=<c>) { ...	for <var> in range(<a>, <b>+1, <c>): ...
for <var> from <a> down to <b> step <c>	for (<var>=<a>; <var><=<b>; <var>-=<c>) { ... }	for <var> in range(<a>, <b>-1, <c>): ...
do ... while <cond>		while True: ... if not <cond>: break
return <value>	return <value>	return <value>
exit	exit(0)	sys.exit(0)
PROCEDIMENTOS, E FUNÇÕES		
procedure <name> (<parameters list>)	void <name> (<parameters list>)	def <name> (<parameters list>)
function <type> <name> (<parameters list>)	<type> <name> (<parameters list>)	def <name> (<parameters list>)

Fonte: elaborado pelo autor.

O módulo tradutor do sistema gerador faz parte do conjunto de peças essenciais para a geração dos exercícios. Ele é o responsável por traduzir uma solução em pseudocódigo para uma solução formatada em uma linguagem de programação definida pelo usuário: *Java*, *C*, *C++* ou *Python*. O módulo tradutor é composto de duas classes: a *TranslateParser* e a *TranslateLexer*. Estas duas

classes são geradas a partir do uso da ferramenta *ANTLR*[2] (*Another Tool for Language Recognition*) (ANTLR, 2016).

Figura 24 - Gramática para o módulo tradutor

```
program → ( statement )*
statement → declare | attribution | read | print | while | if | for
declare → declare type variable newline
attribution → variable = expression newline
read → read variable newline
print → ( print | println ) expression ( , expression )* newline
if → if expression newline indent ( statement ) * dedent ( else newline indent
     ( statement )* dedent )?
while → while expression newline indent ( statement )* dedent
for → for variable from expression ( up | down ) to expression step expression newline
       indent ( statement )* newline
expression → unary_not ( && | || )*
unary_not → ( ! )? exp_comparisson
exp_comparison → exp_arithmetic  ( ( == | != | > | >= | < | <= ) exp_arithmetic )?
exp_arithmetic  → term ( ( + | - ) term )*
term → unary ( ( * | / | % ) unary )*
unary → ( + )? ( - )? factor
factor → number | variable | string | true | false | pi | power ( expression , expression ) |
          random  ( expression , expression ) | ( expression )
type → integer | real | character | boolean | string
```

Fonte: Tramontina, 2015.

Na Figura 24, observa-se as regras da gramática utilizada e criada por Tramontina (2015), propostas para o tradutor na construção do sistema gerador. Para a criação da gramática Tramontina (2015) analisou quais os símbolos do pseudocódigo deveriam ser interpretados.

Nesta parte do sistema gerador, foram adicionadas novas regras no arquivo de gramática, para que o módulo tradutor tenha condições de interpretar os novos comandos e tipos estruturados de variáveis. O módulo tradutor deve identificar que, no momento da leitura do pseudocódigo, caso ele encontrar a palavra *array*, que se refere a um vetor e, desta forma, efetuar o tratamento necessário para criar esse

[2] O *ANTLR* é uma ferramenta que consegue criar analisadores de texto com potencial para ler, processar, traduzir ou executar textos estruturados e até mesmo arquivos binários. Tudo isso se deve a um arquivo de gramática que é usado como parâmetro de entrada para a ferramenta *ANTLR*. A partir da linguagem formal contida no arquivo de gramática, o *ANTLR* gera as classes de *parser* e *lexer*, que serão utilizadas pelo sistema gerador. Todos os comandos detalhados nas figuras 23 e 24 devem estar no arquivo de gramática, para que o módulo tradutor tenha condições de traduzir o comando em pseudocódigo na linguagem de programação especificada.

vetor na linguagem de programação parametrizada. Toda a tradução dos comandos é feita pelas classes *TranslateParser* e *TranslateLexer,* que serão geradas a partir do novo arquivo de gramática enviado a ferramenta *ANTLR,* podendo ser observado na Figura 25.

Figura 25 - Nova gramática para o módulo tradutor

```
program → ( procedure | function )* ( statement )*
procedure → procedure variable ( parameters ) newline indent ( statement )* dedent ( newline )*
function → function type variable ( parameters ) newline indent ( statement )* dedent ( newline )*
parameters → ( declaration ( , declaration )* )?
declaration → ( type variable | array variable of type | matrix variable of type )
type → integer | real | character | boolean | string
statement → declare | attribution | read | print | while | if | for | call | return | newline
declare → declare ( type variable |
                    array variable of type size expression |
                    matrix variable of type size expression by expression ) newline
attribution → variable ( [ exp_arithmetic ] ( [ exp_arithmetic ] )? )? = expression newline
read → read variable newline
print → ( print | println ) expression ( , expression )* newline
while → while exp_comparison newline indent ( statement )* dedent
if → if exp_comparison newline indent ( statement )* dedent ( else newline indent ( statement )* dedent )?
for → for variable from expression ( up | down ) to expression step expression newline
        indent ( statement )* dedent newline
call → variable ( arguments ) newline
arguments → ( expression ( , expression )* )?
return → return expression newline
expression → unary_not ( ( && | || ) unary_not )*
unary_not → ( ! )? exp_comparison
exp_comparison → exp_arithmetic ( ( == | != | < | <= | > | >= ) exp_arithmetic )?
exp_arithmetic → term ( ( + | - ) term )*
term → unary ( ( * | / | % ) unary )*
unary → ( + | - )? factor
factor → number | variable ( [ exp_arithmetic ] ( [ exp_arithmetic ] )? )? | string | true | false | pi |
                    power ( exp_arithmetic , exp_arithmetic ) | random ( exp_arithmetic , exp_arithmetic ) |
                    ( expression ) | variable ( arguments )
```

Fonte: elaborado pelo autor.

Basicamente, a nova gramática desenvolvida adiciona ou altera as seguintes regras:

- **procedure** e **function**: responsáveis pela declaração de funções e procedimentos;
- **parameters**: responsável por interpretar os parâmetros passados às funções e procedimentos;
- **declaration**: responsável por interpretar variáveis simples e estruturadas como parâmetros de entrada das funções e procedimentos;
- **declare**: alterado para que passe a interpretar a declaração de vetores e matrizes;

- **call**: regra responsável por interpretar uma chamada a uma função ou procedimento;
- **arguments**: expressões que podem ser passadas como parâmetros nas chamadas das funções e procedimentos;
- **return**: regra para inserir o *token return* no fim de uma função.

Com as regras de gramática devidamente definidas, será necessário um trabalho de implementação das mesmas no arquivo de gramática, a fim de que os novos recursos sejam interpretados de maneira correta pelo módulo tradutor. No Capítulo seguinte, será detalhado a forma com que os novos recursos foram implementados no módulo tradutor.

5 DESENVOLVIMENTO

Este Capítulo tem como objetivo apresentar os detalhes do desenvolvimento da versão estendida do Gerador Automático de Exercícios de Programação.

5.1 Visão geral

O Gerador Automático de Exercícios de Programação criado por Tramontina (2015) proporcionou aos estudantes de Algoritmos e Programação, uma alternativa ao modelo de aprendizado oferecido pelas instituições de ensino. Porém, a gama de exercícios, do ponto de vista dos conteúdos das disciplinas de Algoritmos e Programação, ainda é limitada pelo fato de que nem todos os comandos utilizados são interpretados pelo sistema gerador. Os novos recursos implementados neste trabalho fazem com que todo o conteúdo estudado nas disciplinas de Algoritmos e Programação seja contemplado, podendo assim, gerar exercícios que utilizem os principais comandos e funções pré-definidas estudados nestas disciplinas.

Para que isso seja possível, o recurso de laços de repetição foi estendido a fim de contemplar o comando *do ... while*. O laço de repetição *do ... while* é um comando utilizado quando um trecho do programa precisa ser executado mais de uma vez. Suas repetições podem estar relacionadas com um número fixo de repetições ou a uma variável do contexto do programa (ASCENCIO, 2012).

Também foram desenvolvidas variáveis estruturadas como vetores e matrizes. Ascencio (2012) define um vetor como uma variável composta unidimensional, ou seja, um conjunto de variáveis de mesmo tipo e de mesmo nome, possuindo um índice para diferenciá-las na estrutura do vetor. Seu armazenamento na memória é feito de forma sequencial. Uma matriz, diferentemente de um vetor, é uma estrutura multidimensional, que possui variáveis de mesmo tipo e de mesmo

nome. O que difere umas das outras são os índices da matriz. Cada dimensão da matriz necessita de um índice (ASCENCIO, 2012).

Os exercícios gerados pelo sistema devem suportar procedimentos e funções, que também podem ser chamadas de sub-rotinas. Uma sub-rotina é um trecho de código que pode ser executado, referenciando pelo seu nome, a qualquer momento do processamento da rotina principal. As sub-rotinas podem ou não receber e devolver variáveis como parâmetros. Para Ascencio (2012, p. 252), a definição de sub-rotina pode ser dada por:

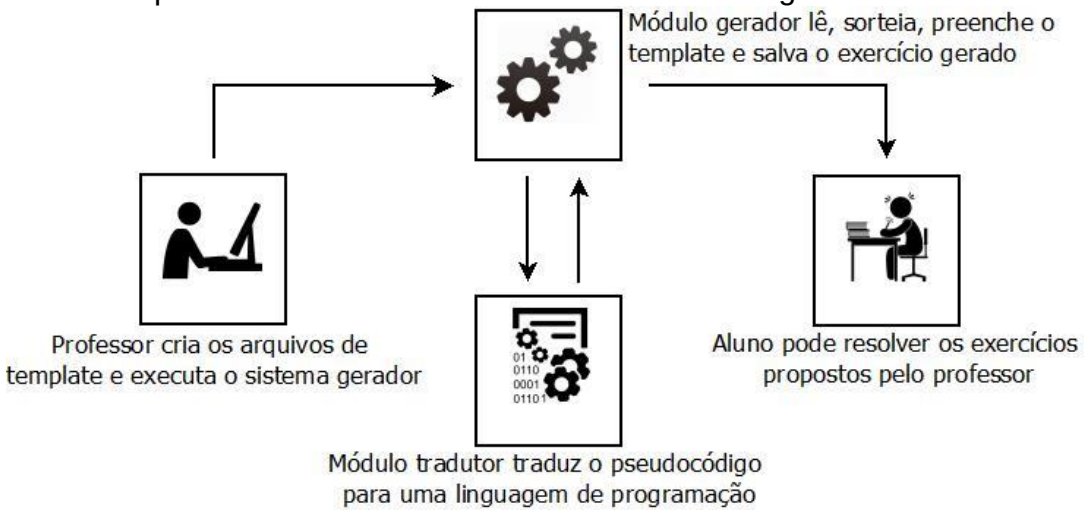
> Sub-rotinas, também chamadas subprogramas, são blocos de instruções que realizam tarefas específicas. O código de uma sub-rotina é carregado uma vez e pode ser executado quantas vezes forem necessárias. Como o problema pode ser subdividido em pequenas tarefas, os programas tendem a ficar menores e mais organizados.

Quando um programa qualquer inicia sua execução, os comandos são executados em sequência. Caso o programa encontre uma chamada a uma sub-rotina, seu fluxo normal de execução é desviado e o código da sub-rotina é executado. Quando a execução da sub-rotina chega ao fim, o programa retorna ao fluxo normal de execução (ASCENCIO, 2012).

Para contemplar todos os conteúdos abordados nas disciplinas de Algoritmos e Programação, o uso dos conceitos de vetores, matrizes, procedimentos e funções são de fundamental importância. Em uma sala de aula, o professor poderá aplicar exercícios desde o início da disciplina, com conteúdo de nível básico, até o fim dela, com conteúdo considerado de nível médio a avançado. Neste caso, o professor cria o repositório com os exercícios utilizando os novos conceitos, e executa o sistema gerador.

O sistema, então, lê o repositório dos exercícios, seleciona os *templates* do nível escolhido pelo professor e o preenche com os parâmetros estabelecidos nos próprios *templates*. Após o preenchimento, o bloco do exercício que contém o pseudocódigo é enviado para o módulo tradutor e este módulo faz com que o programa em pseudocódigo seja traduzido para a linguagem parametrizada pelo professor. Neste momento, os novos conceitos implementados neste trabalho serão interpretados e traduzidos. O módulo tradutor devolve o código para o módulo gerador, que por fim salva o arquivo com a versão final e o ciclo é concluído. Na Figura 26, é possível visualizar com detalhes como este processo ocorre.

Figura 26 - Esquema básico de funcionamento do sistema gerador



Fonte: elaborado pelo autor.

Na Figura 26, apresenta-se uma ideia do funcionamento básico e quais as funções dos módulos do sistema gerador.

O professor é o responsável por criar os *templates* e executar o sistema gerador, via linha de comando. Na linha de comando são passados parâmetros como o número de exercícios, nível de dificuldade, tópicos abordados e também o caminho onde ficarão os arquivos dos exercícios depois de gerados. O módulo gerador lê, sorteia um dos *templates* e o preenche com dados aleatórios. O módulo tradutor lê o pseudocódigo do *template* selecionado e faz a tradução do mesmo para a linguagem de programação parametrizada pelo usuário. Por fim, o módulo gerador faz a gravação do arquivo no diretório da lista de exercícios.

Com estes recursos estendidos será possível alterar o arquivo do pseudocódigo para que gere exercícios que aborde estes conceitos.

Com o esquema de funcionamento do sistema gerador detalhado, o próximo passo foi analisar os requisitos dos novos recursos a serem implementados no módulo tradutor. Os requisitos e sua função podem são observados na seção a seguir.

5.2 Análise de Requisitos

Esta seção tem o propósito de analisar e detalhar os requisitos necessários para o desenvolvimento dos recursos a serem estendidos do gerador de exercícios.

5.2.1 Requisitos Funcionais

Tabela 1 - Requisito Funcional 1

Requisito	RF001
Nome	Permitir o uso do comando *do ... while*
Descrição	
Será possível criar *templates* de exercícios, utilizando o comando *do ... while* como alternativa no recurso de laços de repetição.	

Fonte: elaborado pelo autor.

Tabela 2 - Requisito Funcional 2

Requisito	RF002
Nome	Possibilitar o uso de vetores
Descrição	
Será possível criar *templates* de exercícios, utilizando variáveis estruturadas, como vetores.	

Fonte: elaborado pelo autor.

Tabela 3 - Requisito Funcional 3

Requisito	RF003
Nome	Permitir o uso de matrizes
Descrição	
Será possível criar *templates* de exercícios, utilizando variáveis estruturadas, como matrizes.	

Fonte: elaborado pelo autor.

Tabela 4 - Requisito Funcional 4

Requisito	RF004
Nome	Permitir o uso procedimentos
Descrição	
Será possível criar *templates* de exercícios, utilizando procedimentos como sub-rotinas da rotina principal.	

Fonte: elaborado pelo autor.

Tabela 5 - Requisito Funcional 5

Requisito	RF005
Nome	Permitir o uso funções
Descrição	
Será possível criar *templates* de exercícios, utilizando funções como sub-rotinas da rotina principal.	

Fonte: elaborado pelo autor.

Tabela 6 - Requisito Funcional 6

Requisito	RF006
Nome	Permitir o uso da função *length*
Descrição	
Será possível criar *templates* de exercícios, utilizando a função *length* para obter o tamanho de variáveis do tipo *string* e de vetores e matrizes.	

Fonte: elaborado pelo autor.

5.2.2 Requisitos Não Funcionais

Tabela 7 - Requisito Não Funcional 1

Requisito	RNF001
Nome	Implementar os recursos estendidos em *Java*
Descrição	
O desenvolvimento dos recursos deverá utilizar a linguagem de programação *Java*, pois é a mesma linguagem do desenvolvimento do Gerador de Exercícios.	

Fonte: elaborado pelo autor.

Tabela 8 - Requisito Não Funcional 2

Requisito	RNF002
Nome	Padronizar os exercícios no formato de pseudocódigo
Descrição	
Os *templates* dos exercícios propostos deverão seguir um padrão em formato de pseudocódigo.	

Fonte: elaborado pelo autor.

Tabela 9 - Requisito Não Funcional 3

Requisito	RNF003
Nome	Executar a geração dos exercícios através de linha de comando
Descrição	
A ação de execução de geração dos exercícios deverá ocorrer via linha de comando.	

Fonte: elaborado pelo autor.

Tabela 10 - Requisito Não Funcional 4

Requisito	RNF004
Nome	Utilizar a ferramenta *ANTLR* para a tradução
Descrição	
A tradução do pseudocódigo para as linguagens de programação deverá ser feita pelas classes *TranslateLexer* e *TranslateParser,* geradas pela ferramenta *ANTLR*.	

Fonte: elaborado pelo autor.

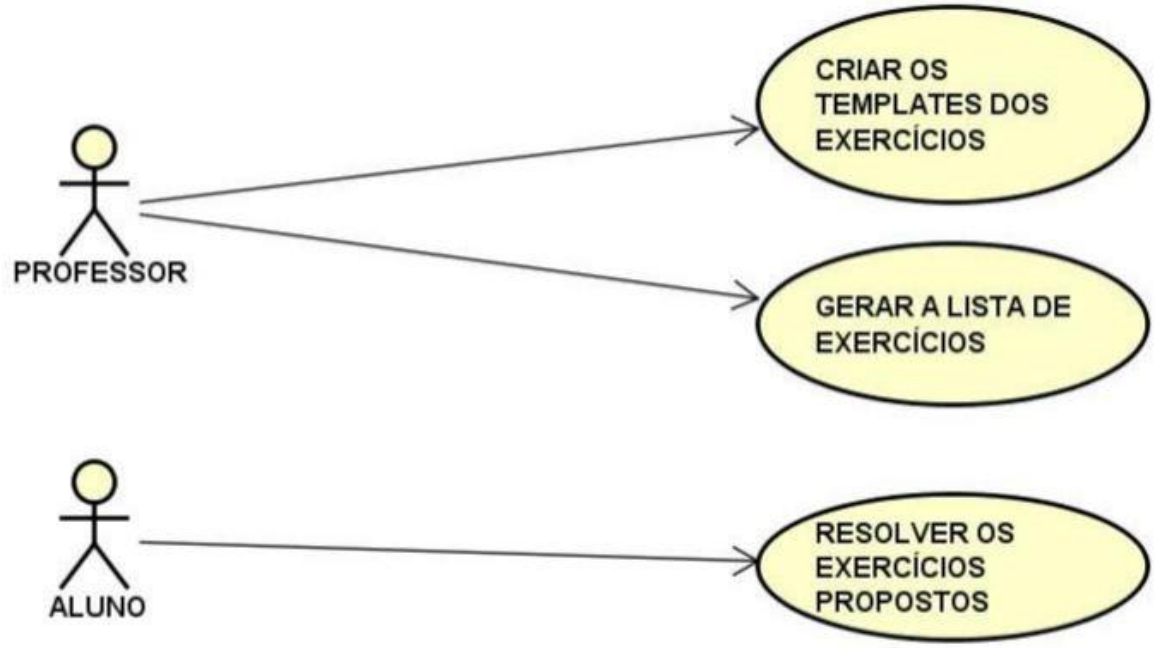
5.3 Diagramas Comportamentais

Esta seção pretende detalhar os diagramas comportamentais necessários para o desenvolvimento dos novos recursos do gerador de exercícios. O professor tem papel fundamental na geração dos exercícios, pois ele precisa criar o repositório, onde estarão os *templates* dos exercícios e gerá-los no sistema gerador. O aluno fica somente com o papel de resolver os exercícios gerados.

5.3.1 Diagrama de casos de uso

Como pode ser observado na Figura 27, o professor tem um papel fundamental na utilização do sistema Gerador, pois ele tem por objetivo criar os arquivos de template, que servirão de base para os exercícios, e também disparar o processo de geração via linha de comando. Só assim, os alunos poderão resolver as atividades propostas pelo professor e geradas pelo sistema.

Figura 27 - Diagrama de casos de uso



Fonte: elaborado pelo autor.

A Figura 27 detalha o diagrama de casos de uso. Nos papéis de usuários, estão os professores e os alunos. Os professores terão a tarefa de criar os arquivos de *templates* dos exercícios e executar o sistema gerador. Os alunos terão o papel de resolver os exercícios propostos pelos professores, gerados de maneira automática pelo sistema gerador. Nas tabelas 13, 14 e 15, observa-se o detalhamento dos casos de uso.

Tabela 11 - Descrição do caso de uso “criar os templates dos exercícios”

Caso de uso	UC01
Nome	Criar os *templates* dos exercícios
Descrição	
Os professores deverão criar os arquivos de *template* que servirão de base para que a lista de exercícios possa ser gerada e aplicada junto aos alunos.	

Fonte: elaborado pelo autor.

Tabela 12 - Descrição do caso de uso “gerar a lista de exercícios”

Caso de uso	UC02
Nome	Gerar a lista de exercícios
Descrição	
Com os arquivos de *template* elaborados, o professor poderá executar o sistema gerador para que a lista de exercícios seja gerada.	

Fonte: elaborado pelo autor.

Tabela 13 - Descrição do caso de uso “resolver os exercícios propostos”

Caso de uso	UC03
Nome	Resolver os exercícios propostos
Descrição	
Os alunos poderão resolver os exercícios propostos pelo professor, que foram gerados pelo sistema gerador.	

Fonte: elaborado pelo autor.

Superada a etapa de definição dos papéis dos usuários, foi necessário analisar onde e como as implementações seriam realizadas. Uma análise prévia mostrou que o módulo que sofreria mais impacto seria o tradutor. Fazendo o uso da ferramenta *ANTLR*, o arquivo de gramática que serve como parâmetro de entrada desta ferramenta precisou contemplar os novos recursos, através de regras criadas para interpretar os novos comandos. A seção a seguir, mostra com maior detalhamento a forma com que as regras foram implementadas no arquivo de gramática.

5.4 Implementação dos recursos estendidos do sistema gerador

A implementação dos novos recursos do sistema gerador ocorreu de forma mais significativa no seu módulo de tradução. Utilizando a ferramenta *ANTLR*, o módulo tradutor será capaz de interpretar os símbolos necessários para a utilização de laços de repetição com o comando *do ... while* e para a declaração, não só de vetores e matrizes, mas também de funções, procedimentos e outras funções

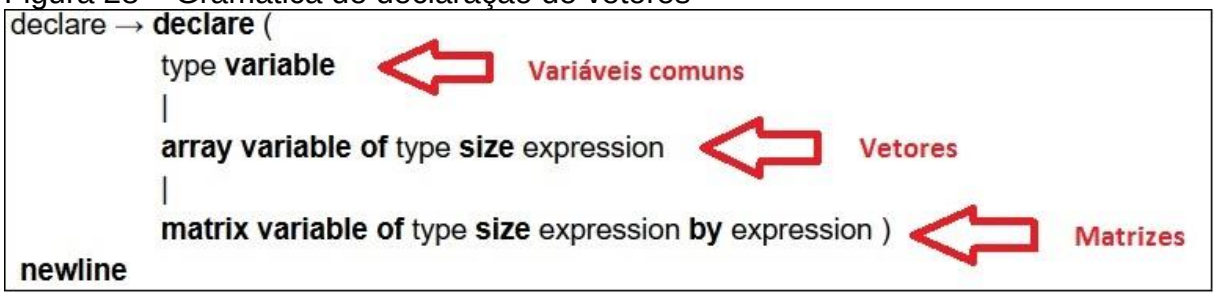
predefinidas. O módulo tradutor recebe o bloco de texto do *template* do exercício que contém o pseudocódigo e o lê sequencialmente. Durante a esta leitura, o módulo tradutor vai interpretando os símbolos e aplicando as regras, baseado no arquivo de gramática com as novas regras implementadas.

Com as novas regras que foram implementadas, o módulo de tradução é capaz de interpretar quando vetores ou matrizes são declarados, assim como quando o pseudocódigo possuir algum procedimento ou função. Nas seções seguintes, pode-se ver, detalhadamente, como cada regra foi implementada, na declaração de vetores, matrizes, procedimentos, funções, uso da função predefinida *length* e uso de laço de repetição com o comando *do ... while*.

5.5 Interpretação de vetores e matrizes

Para possibilitar o uso de vetores e matrizes, foi preciso alterar as regras da gramática em três pontos básicos: declaração, atribuição e uso. A declaração dos vetores e matrizes são bem semelhantes e seguem a sintaxe *declare array* <nome> *of integer size* <tamanho>. Na Figura 28, podem-se ver as regras da gramática para a declaração das variáveis.

Figura 28 – Gramática de declaração de vetores



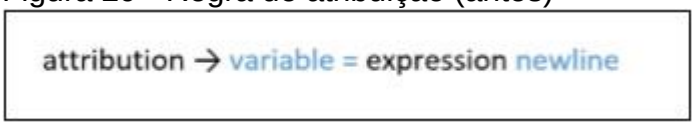
Fonte: elaborado pelo autor.

Na Figura 28, as palavras grifadas são os *tokens*. A regra tem início pelo *token* obrigatório ***declare***. As regras entre parênteses que estão separadas pelo caractere “|” permite a aplicação de uma ou outra destas regras. Por exemplo, para a declaração de variáveis comuns (ex: *declare integer* i), o módulo tradutor irá validar que a primeira regra é a válida (***declare*** *type* ***variable***), já que a sintaxe do pseudocódigo se encaixa com a estabelecida no módulo tradutor. A regra de declaração de variáveis comuns já tinha sido previamente criada por Tramontina (2015). Ciente da necessidade de declarar vetores e matrizes, foram criadas as outras duas regras para contemplar as declarações. Sempre será necessária uma

nova linha após cada declaração. Esta obrigatoriedade se deve ao *token* ***newline*** que se encontra no final da regra.

A regra de atribuição das variáveis foi igualmente alterada para contemplar os vetores e matrizes. Na Figura 29, observa-se como Tramontina (2015) implementou a regra de interpretação das atribuições em variáveis comuns.

Figura 29 - Regra de atribuição (antes)

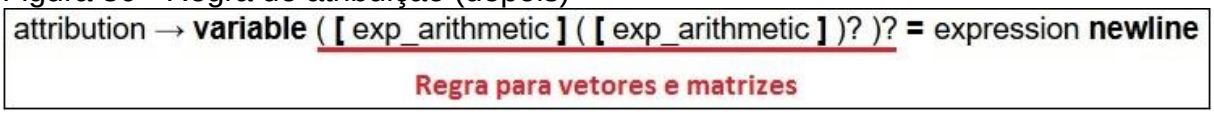


Fonte: Tramontina (2015).

A regra da Figura 29 mostra que o módulo tradutor é capaz de interpretar a sintaxe do pseudocódigo, por exemplo, i = 2. A regra mostra que a sequência de símbolos obrigatória para ocorrer uma atribuição, tem que ser: nome da variável, seguido do caractere “=” (igual), seguido de uma expressão e finalizando com uma nova linha. A expressão pode ser um número, outra variável de mesmo tipo, operação matemática, entre outros.

Para contemplar a atribuição de vetores e matrizes foram adicionadas algumas regras logo após o nome da variável, onde pode haver o caractere “[“, seguido de uma expressão e terminando com o caractere “]”. Isso faz referência ao acesso a uma posição do vetor. Se a atribuição for em uma matriz, logo após o caractere “]”, segue o caractere “[“ seguido de uma expressão e seguido do caractere “]”. Isso faz referência ao acesso à linha e coluna da matriz. As regras de atribuição de vetores e matrizes ficam entre parênteses, seguido do caractere “?”. Isso mostra que para uma atribuição ocorrer, estes trechos não são obrigatórios, como pode ser visto na Figura 30.

Figura 30 - Regra de atribuição (depois)



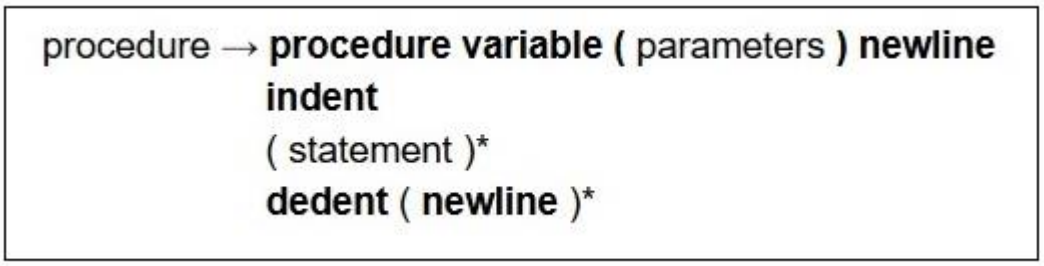
Fonte: elaborado pelo autor.

5.6 Interpretação de procedimentos e funções

Para interpretar procedimentos e funções, foram criadas regras para a declaração e para as chamadas dos mesmos. Na declaração dos procedimentos, foi criada a regra para seguir a sintaxe *procedure* <nome> (<parâmetros de entrada>)

<nova linha> <endentação de quatro espaços> <bloco de código do procedimento> <recuo de quatro espaços> <uma ou mais linhas em branco>. No arquivo de gramática esta regra recebeu o nome de *procedure* e pode ser observada na Figura 31.

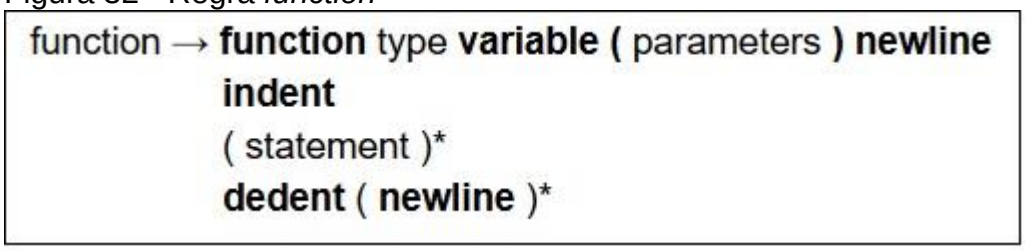
Figura 31 - Regra *procedure*



Fonte: elaborado pelo autor.

As funções também necessitaram de uma regra para sua declaração. Ela é muito semelhante à regra dos procedimentos, a não ser pela palavra *function* ao invés de *procedure* e pelo tipo do dado do retorno após a palavra *function*. Sua sintaxe é *function* <tipo do retorno> <nome da função> (<parâmetros de entrada>) <nova linha> <endentação de quatro espaços> <bloco de código da função> <recuo de quatro espaços> <uma ou mais linhas em branco>. No arquivo de gramática esta regra recebeu o nome de *function* e pode ser observada na Figura 32.

Figura 32 - Regra *function*



Fonte: elaborado pelo autor.

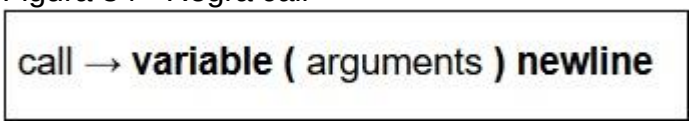
A regra *function* ainda conta com um símbolo de retorno chamado *return*. Este símbolo informa qual o dado que será retornado pela função. Sua sintaxe pode ser observada na Figura 33.

Figura 33 - Regra *return*

Fonte: elaborado pelo autor.

A regra do *return* mostra que não precisa retornar necessariamente uma variável, mas qualquer tipo de expressão. A chamada tanto das funções como dos procedimentos foi implementada na regra *call*. A regra call pode ser observara na Figura 34.

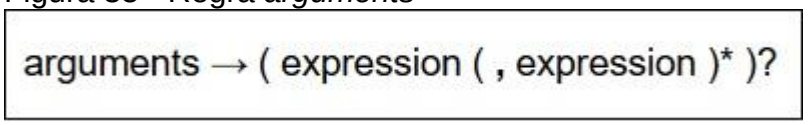
Figura 34 - Regra *call*



Fonte: elaborado pelo autor.

A Figura 34 mostra que a sintaxe da regra *call* é <nome do procedimento / função> (<argumentos>) <nova linha>. Para os argumentos, foi criada a regra *arguments*, que pode ser uma expressão ou várias expressões, mas separadas por vírgulas, como pode ser observado na Figura 35.

Figura 35 - Regra *arguments*



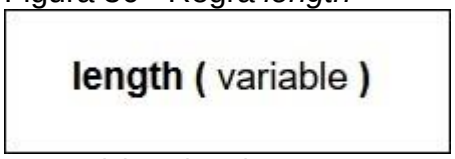
Fonte: elaborado pelo autor.

Além da criação de regras para a interpretação de procedimentos e funções, foi criada uma regra para interpretar uma função auxiliar que é muito utilizada nas disciplinas de Algoritmos e Programação, que é a função *length*. O detalhamento da função *length* pode ser observado na seção a seguir.

5.7 Interpretação da função *length*

A função length é utilizada para obter o tamanho do dado armazenado nas variáveis. No arquivo de gramática, a função *length* seguiu a mesma ideia de estrutura das funções predefinidas já existentes, como a *random* e a *pow*. Sua sintaxe em pseudocódigo segue a regra *length* (<variável>) e pode ser observada na Figura 36.

Figura 36 - Regra *length*



Fonte: elaborado pelo autor.

Com a função *length*, é possível obter os tamanhos de variáveis do tipo *string*, vetores e matrizes. Para obter o tamanho das variáveis em *Java*, foi utilizada a função *length*() e para os vetores, foi utilizado o atributo *length*. Nas linguagens *C* e *C++*, para as variáveis *string* foi utilizada a função *strlen*() e para os vetores, foi criada uma variável interna que se encarrega de fazer o controle do tamanho do vetor. Esta variável interna tem seu nome no padrão <*nome do vetor*>*_len*. Pelo fato desta variável utilizar o caractere sublinhado (_), ela não conflitará com nenhuma variável criada pelo usuário no pseudocódigo, pois não é permitido declarar variáveis que utilizam este caractere.

Para obter o tamanho das matrizes, foram implementadas duas funções predefinidas no pseudocódigo: a função *rows* e a função *columns*, que significam linhas e colunas respectivamente. Estas funções recebem a matriz como parâmetro e devolvem o tamanho de linhas ou colunas. Na linguagem *Java*, foi utilizado o atributo *length*. Quando este atributo é utilizado da forma <*matriz*>*.length*, o número que irá retornar será o número de linhas. Para obter o número de colunas da matriz, o módulo tradutor gerará o código <*matriz*>*[0].length*. Para as linguagens *C* e *C++*, foi utilizada a ideia de variáveis auxiliares para fazer este controle. Neste caso, para as matrizes, duas variáveis são criadas: a <*matriz*>*_row* e a <*matriz*>*_col*, que armazenarão o número de linhas e colunas respectivamente.

Tanto para obter o tamanho de *strings*, ou para vetores ou matrizes, na linguagem *Python* foi utilizado a função *len*(). A função *len*() recebe como parâmetro a variável do tipo *string*, ou o vetor ou a matriz. Para as matrizes, para obter o número de linhas, o módulo tradutor gerará o código *len*(<matriz>), enquanto para obter o número de colunas, o módulo tradutor gerará o código *len(<matriz>[0])*.

5.8 Interpretação da função *do ... while*

A interpretação do comando *do ... while* foi implementada no arquivo de gramática, através de regras, semelhantes a dos comandos *while* e *for*. A sintaxe segue o padrão *do* <nova linha> <endentação de 4 espaços> <bloco com código>

<recuo de 4 espaços> *while* <expressão de comparação>, como pode ser observado na Figura 37.

Figura 37 - Regra *do ... while*

do → **do newline indent** (statement)* **dedent while** exp_comparison

Fonte: elaborado pelo autor.

A linguagem *Python* não possui o laço de repetição *do ... while* como a linguagem *Java*, *C* e *C++*, então, quando o programa em pseudocódigo for traduzido para *Python* possuir um laço de repetição *do ... while*, o módulo tradutor gerará um laço de repetição com o comando *while*, com a expressão de comparação verdadeira (*true*), e no fim do laço, existirá um comando *if*, testando a expressão de comparação original invertida, e se o resultado deste teste for verdadeiro, o programa executará o comando *break*, o que faz com que a execução prossiga com os comandos após o laço de repetição. Esta adaptação pode ser observada na Figura 38.

Figura 38 - Adaptação *do ... while* para Python

```
z = 3                          z = 3
y = 0                          y = 0

do                             while True:
    x = 2                          x = 2
    y = y + x + 2                  y = y+x+2

    if y == 8                      if y == 8:
        z = 4                          z = 4

while z == 3                       if z != 3:
                                       break
```

PSEUDOCÓDIGO PYTHON

Fonte: elaborado pelo autor.

Com os recursos devidamente implementados, o próximo passo foi a criação do repositório com os *templates* dos exercícios. Estes *templates* serviram para pôr em teste o sistema gerador em um cenário real. O cenário escolhido para a aplicação dos testes foram os professores das disciplinas de Algoritmos e Programação, onde a ferramenta será disponibilizada. Os professores fizeram o teste de gerar listas de exercícios para a linguagem *Java*. Após o teste, foi aplicado

um questionário, no qual os professores puderam sinalizar se tiveram alguma dificuldade no processo ou se acharam que a ferramenta é válida para ser utilizada nas disciplinas, como pode ser observado no Capítulo 5.

6 AVALIAÇÃO E RESULTADOS OBTIDOS

Após a conclusão das implementações propostas no presente estudo, foi realizada uma avaliação da ferramenta junto aos professores que ministram as disciplinas de Algoritmos e Programação. Esta avaliação objetivou mensurar se o sistema gerador é, de fato, uma ferramenta que poderá ajudar o aluno a assimilar de forma mais significativa os conteúdos estudados na disciplina. Antes do teste ser aplicado foi necessário criar um repositório de *templates* capaz de gerar os exercícios.

Este repositório é composto de 4 *templates* com exercícios sobre vetores, 2 sobre matrizes, 2 sobre funções e 1 sobre procedimentos, totalizando 9 *templates* de exercícios. Os *templates* criados têm como base listas de exercícios utilizadas pelos alunos na prática (APÊNDICE B, APÊNDICE C, APÊNDICE D e APÊNDICE E).

Para o experimento, foi disponibilizado um diretório contendo o protótipo do sistema Gerador, o repositório com os 9 *templates*, o instalador do software *MarkdownPad*[3], um arquivo executável *.bat* com a linha de comando e os parâmetros para executar o sistema Gerador, visando dinamizar a geração dos exercícios e o diretório, onde serão gravados os exercícios gerados.

Além do diretório, foi elaborado um manual do gerador automático de exercícios (APÊNDICE F), com o objetivo de esclarecer algumas dúvidas e detalhar cada componente do diretório, bem como a linha de comando com os parâmetros necessários para disparar o processo de geração.

Foram selecionadas 10 pessoas para participar do experimento. Todos são professores, ou tem contato, com os alunos das disciplinas de Algoritmos e

[3] *MarkdownPad* é um software utilizado para visualizar e editar arquivos que utilizam a linguagem de marcação *Markdown*. Estes arquivos possuem a extensão *.md*.

Programação. Destes 10 professores, 4 aceitaram participar do experimento, totalizando 40% dos professores selecionados.

Neste primeiro momento, o sistema Gerador será utilizado somente pelos professores, pois o mesmo ainda está na fase de prototipação e não possui interface gráfica, deixando o processo de geração de forma manual.

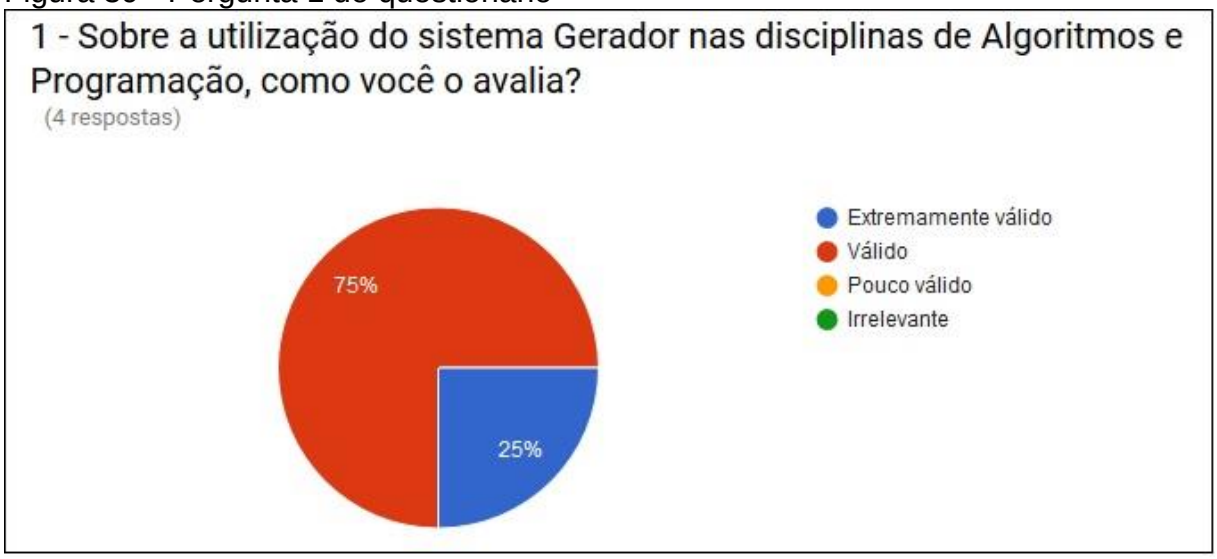
Foi elaborado um e-mail para os avaliadores, explicando o trabalho e seus objetivos de forma sucinta. Também foi solicitado o desenvolvimento do experimento, baseado nas informações disponíveis no Manual do Gerador Automático de Exercícios (APÊNDICE F). Este manual detalha todos os recursos do diretório disponibilizado e também explica como gerar os exercícios utilizando a ferramenta.

Mesmo com o diretório de *templates* disponível para a geração, foi solicitado no e-mail a elaboração de pelo menos um *template*, para que tivessem a experiência de criar um exercício desde o início. Por fim, me coloquei à disposição para esclarecer as dúvidas e auxiliar com as dificuldades encontradas no desenvolvimento do experimento. Dos professores que avaliaram a ferramenta, nenhum deles teve dificuldades a ponto de retornar o e-mail solicitando auxílio.

Ao finalizar o experimento, os avaliadores foram submetidos ao questionário de satisfação do gerador automático de exercícios (APÊNDICE A). Este questionário teve como objetivo coletar informações de *feedback* dos avaliadores do experimento. Para este processo foi escolhida a aplicação de uma pesquisa quantitativa, pois este tipo de pesquisa nos permite conhecer em números a experiência que os avaliadores tiveram ao experimentar a ferramenta. As questões do questionário foram elaboradas de forma objetiva, visando observar se o avaliador teve dificuldades ou se acha complexo o processo de geração dos exercícios.

Na primeira pergunta, como pode ser observado na Figura 39, foi questionado aos avaliadores se eles achavam válida a utilização do sistema Gerador nas aulas das disciplinas de Algoritmos e Programação. Do total de 4 respostas, 3 deles responderam que o uso sistema Gerador nas disciplinas de Algoritmos e Programação é válido, enquanto 1 respondeu que a utilização é extremamente válida, totalizando 75% e 25% respectivamente.

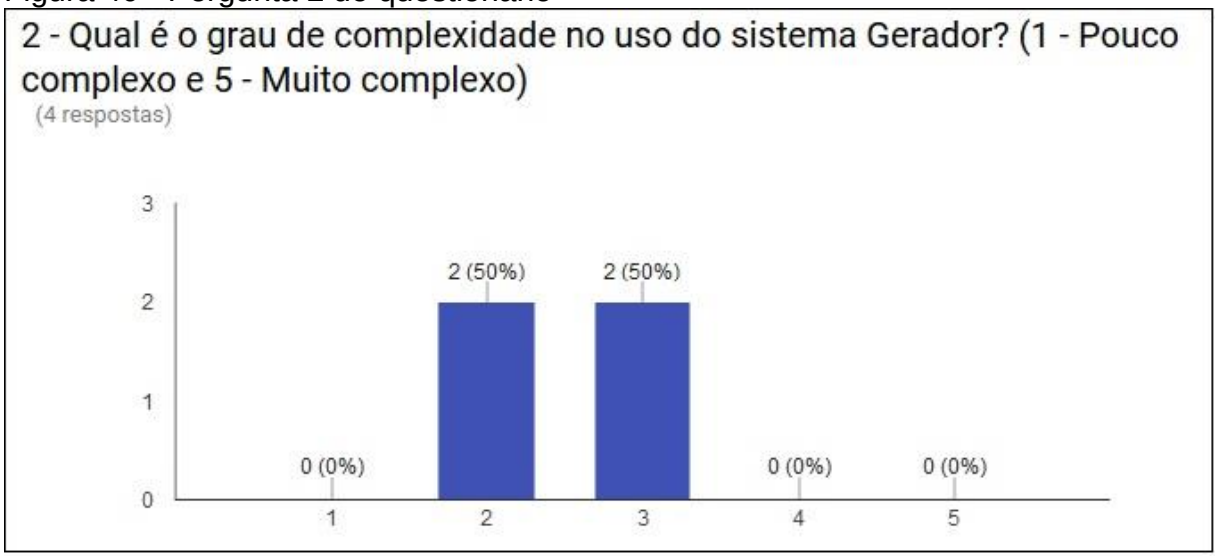
Figura 39 - Pergunta 1 do questionário



Fonte: elaborado pelo autor.

Na segunda pergunta, como pode ser observado na Figura 40, foi questionado aos avaliadores, em uma escala de 1 a 5, onde 1 é pouco complexo e 5 muito complexo, qual o grau de complexidade do uso do sistema Gerador. Nesta pergunta os avaliadores se dividiram, mostrando que a experiência com o sistema Gerador teve uma complexidade média a baixa, totalizando dois avaliadores que responderam complexidade 2 e dois avaliadores que responderam complexidade 3.

Figura 40 - Pergunta 2 do questionário

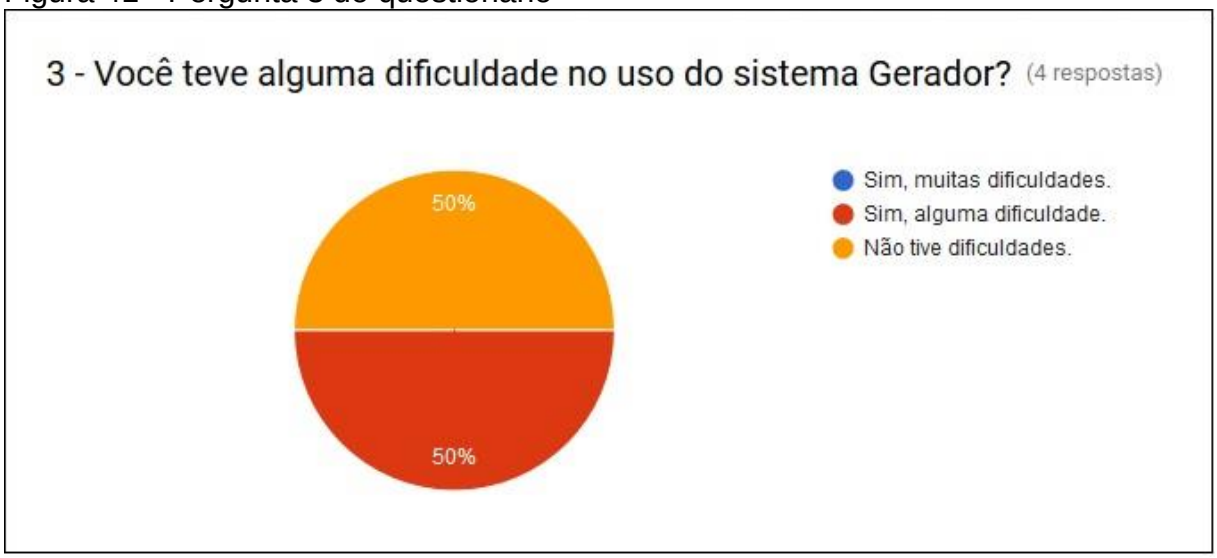


Fonte: elaborado pelo autor.

Na terceira pergunta, como pode ser observado na Figura 41, foi questionado se os avaliadores tiveram alguma dificuldade no uso do sistema Gerador. Esta

questão ficou dividida entre dois avaliadores que não tiveram dificuldade e dois que tiveram alguma dificuldade no experimento.

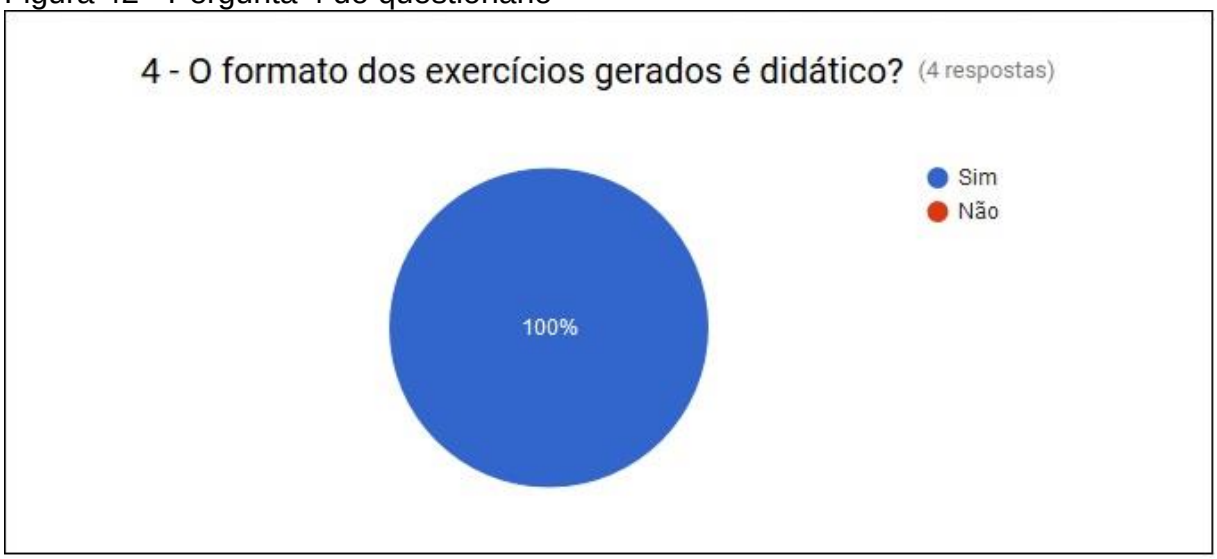
Figura 41 - Pergunta 3 do questionário



Fonte: elaborado pelo autor.

A quarta pergunta, que pode ser vista na Figura 42, foi sobre o formato dos exercícios. Foi questionado aos avaliadores se eles achavam que o formato dos exercícios é didático. Por unanimidade, todos eles responderam que é didático.

Figura 42 - Pergunta 4 do questionário

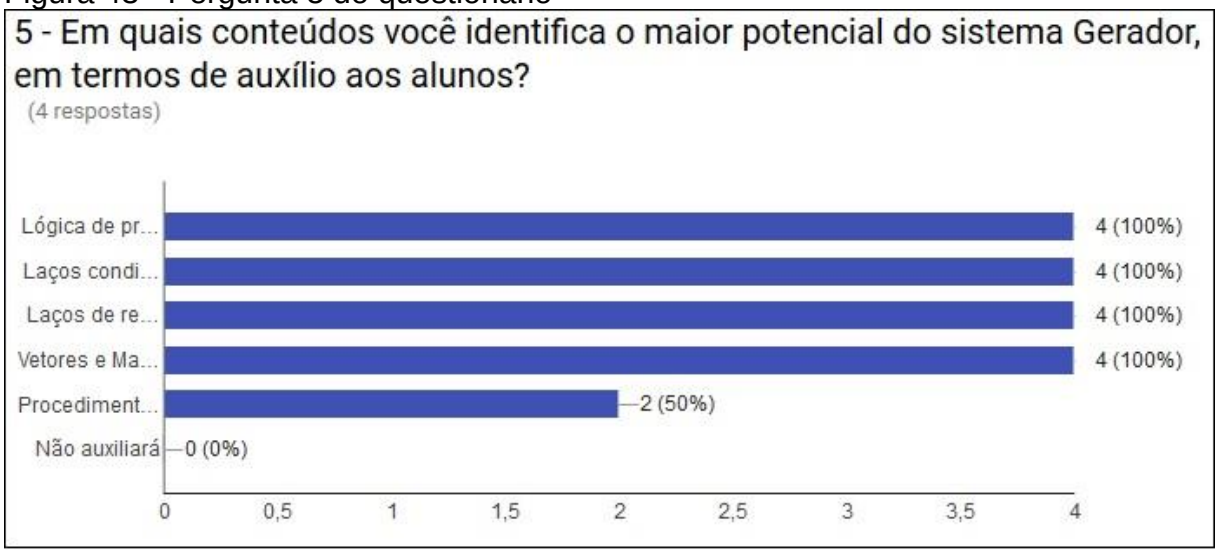


Fonte: elaborado pelo autor.

Na quinta pergunta, Figura 43, foi solicitado aos avaliadores em quais conteúdos eles identificavam um maior potencial do sistema Gerador, no sentido de

auxílio aos alunos. Todos os avaliadores responderam que o sistema Gerador tem grande potencial em auxiliar os alunos nos conteúdos de lógica de programação, laços condicionais *if*, laços de repetição *while* e *for*, vetores e matrizes, totalizando 100% dos avaliadores. Somente dois avaliadores responderam que o sistema Gerador tem grande potencial de auxiliar os alunos nos conteúdos de procedimentos e funções, totalizando 50% dos avaliadores.

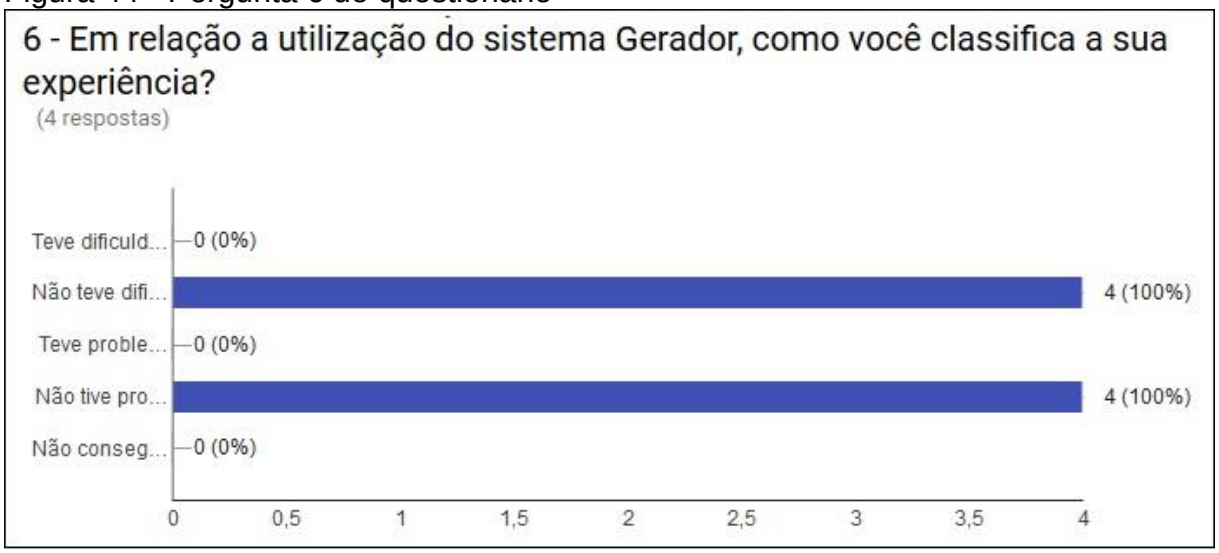
Figura 43 - Pergunta 5 do questionário



Fonte: elaborado pelo autor.

A sexta pergunta, como se observa na Figura 44, foi solicitado aos avaliadores como eles classificam a experiência de utilização do sistema Gerador. Todos eles responderam que não tiveram dificuldades em entender como a ferramenta funciona e também que não tiveram problemas em entender como os *templates* funcionam, totalizando 100% dos avaliadores.

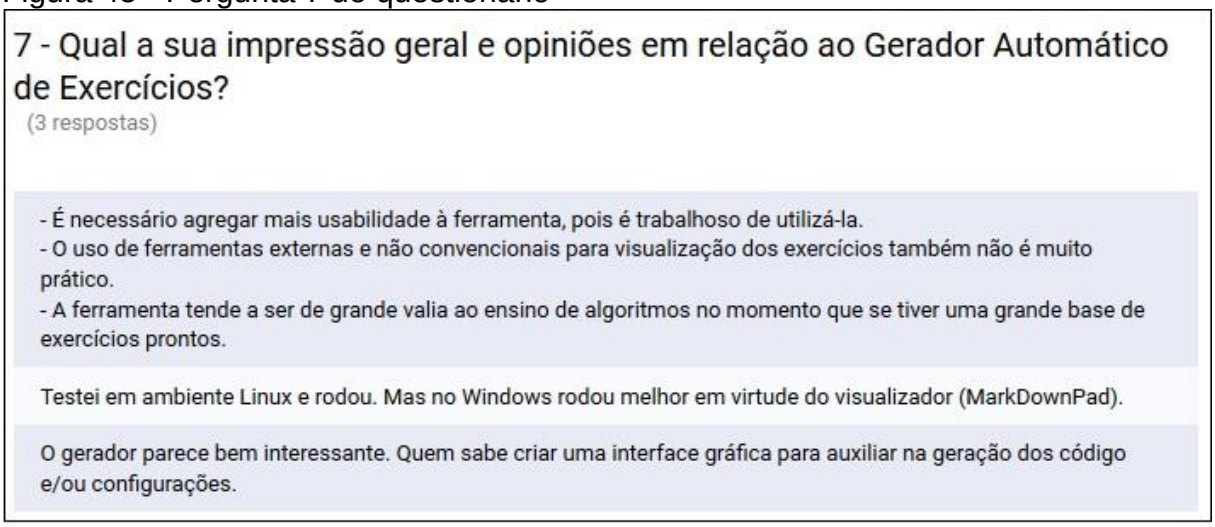
Figura 44 - Pergunta 6 do questionário



Fonte: elaborado pelo autor.

A sétima e última pergunta, representada na Figura 45, foi uma pergunta dissertativa, não obrigatória, onde foi questionado aos avaliadores qual foi a impressão geral e opiniões sobre o sistema Gerador. Dos quatro avaliadores, um optou por não responder esta pergunta, totalizando 75% dos avaliadores. Nesta pergunta, os avaliadores deram suas impressões sobre a ferramenta e também deram sugestões de melhorias, do tipo que seria interessante desenvolver uma interface gráfica para auxiliar na geração dos códigos e/ou configurações. Um dos avaliadores relatou que fez o teste na plataforma Linux e o sistema funcionou. Isso se deve ao sistema Gerador e suas bibliotecas serem escritas na linguagem *Java*, que é multiplataforma. O outro avaliador respondeu que o sistema Gerador será de grande importância para a aprendizagem de Algoritmos e Programação quando tiver uma grande base de *templates* de exercícios prontos. Ele também relatou que seria interessante agregar mais usabilidade à ferramenta, pois ainda é trabalhoso utilizá-la.

Figura 45 - Pergunta 7 do questionário



Fonte: elaborado pelo autor.

Com este experimento, foi possível simular casos reais de uso do sistema Gerador. Este experimento dá uma noção do quanto o sistema gerador pode auxiliar o desenvolvimento do conhecimento dos alunos de Algoritmos e Programação, como também identificar pontos onde o sistema deve ser melhorado para cada vez mais suprir a necessidade os futuros usuários desta ferramenta.

7 CONSIDERAÇÕES FINAIS

O presente trabalho detalhou um conjunto de ferramentas que auxiliam o processo de aprendizagem de programação. Neste estudo foi observado que a grande maioria das ferramentas utilizam uma lista estática de exercícios e, com o passar do tempo de uso, pode resultar na cessão da aprendizagem do aluno, que não aprenderá nada de novo, além do que está sendo mostrado. Ciente desta carência foi desenvolvido um sistema gerador automático de exercícios de programação. Cabe ressaltar que, nas pesquisas realizadas no presente trabalho, não foi encontrada nenhuma ferramenta que gere exercícios de forma automática e que tenha a possibilidade de gerar exercícios em quatro linguagens de programação e Português estruturado. Contudo, o mesmo não atendia a necessidade por completo, pois não era possível gerar exercícios com todos os recursos básicos abordados na disciplina de Algoritmos e Programação, como laços de repetição com *do ... while*, vetores e matrizes, procedimentos, funções e funções predefinidas, como por exemplo a função *length,* utilizada para obter o tamanho de *Strings*. Neste cenário, o presente trabalho se propôs a estender o sistema Gerador, implementando novos recursos, para que os exercícios pudessem ser aplicados em relação a todos os assuntos abordados na disciplina, independentemente do nível de conhecimento dos alunos.

Este trabalho teve por objetivo estender recursos do sistema Gerador previamente desenvolvido. Dos objetivos específicos propostos neste trabalho, as implementações dos conceitos de vetores, matrizes, procedimentos, funções e a validação da ferramenta junto aos professores foram alcançados com êxito. Dentre os objetivos específicos que não foram alcançados com êxito, estão a ampliação do conjunto de operações matemáticas e o dicionário para tradução dos exercícios para

o Português. Para que os objetivos fossem alcançados, foi necessário realizar um estudo nas ferramentas e técnicas existentes no mercado juntamente com seus conceitos. Com este estudo realizado, foi possível determinar quais partes do sistema Gerador sofreriam alterações em função das novas implementações e se constatou que a maior parte dessas novas implementações ocorreriam no módulo tradutor. A base do módulo tradutor é um arquivo de gramática, o qual possui regras léxicas de sintaxe do pseudocódigo utilizado nos templates do sistema Gerador. Estas regras tiveram que sofrer alterações e também foi necessário criar novas regras para suportar a sintaxe e interpretação dos símbolos dos novos recursos que foram implementados. Esta parte foi a mais delicada e que demandou a maior parte do tempo gasto, pois as alterações e criação de novas regras passam por um estudo específico da disciplina de Compiladores. O arquivo de gramática com as novas regras foi submetido à biblioteca do *ANTLR*, que por sua vez gerou as classes necessárias para serem acopladas no sistema Gerador. Por outro lado, as implementações de funções matemáticas predefinidas e um dicionário de tradução dos exercícios para o português, descritas nos objetivos específicos, não foram implementados, em função de demandar uma análise de impacto mais aprofundada nas quatro linguagens de programação que o sistema Gerador atende. Cabe reforçar que qualquer pseudocódigo escrito, obedecendo as regras de sintaxe estabelecidas, poderá ser traduzido para até quatro linguagens de programação como, *C*, *C*++, *Python* e *Java*, como também para o Português estruturado, já incluindo os novos recursos estendidos no presente trabalho.

Num primeiro momento o sistema Gerador terá seu uso destinado aos professores das disciplinas de Algoritmos e Programação, pois estes serão os responsáveis por criar os arquivos de *template* utilizados na geração dos exercícios. A criação dos arquivos de *template* tem de passar por um processo manual, onde o professor necessita escrever o *template*. O processo de geração dos exercícios também precisa ser disparado de forma manual, via linha de comando e, por este motivo, optou-se por deixar somente os professores como usuários principais do sistema. Neste processo foram selecionados professores para um experimento com a ferramenta, os quais foram submetidos a uma pesquisa quantitativa, a fim de avaliar a experiência de cada um, utilizando a ferramenta. Com o resultado desta pesquisa foi possível observar que os objetivos deste trabalho foram alcançados, pois os professores passaram pela experiência desde a criação dos arquivos de

template, já com os novos recursos, até a geração dos exercícios finais, aprovando as funcionalidades da ferramenta. Na última pergunta do questionário, os professores deixaram suas impressões sobre a experiência vivenciada, e como sugestão, um deles relatou que poderia ser desenvolvido uma interface gráfica para a criação dos arquivos de *templates*. Esta pode ser uma melhoria a ser implementada em um próximo trabalho, como também, implementações de funções matemáticas predefinidas como *sqrt*, funções para obter partes de palavras ou frases como o *substring*, e também um banco de dados, onde seriam armazenados os templates para a geração dos exercícios.

Cabe salientar que este trabalho é uma evolução de um sistema previamente desenvolvido e que ainda há muito para evoluir. Esta ferramenta pode ser de grande valia para o aprendizado de programação e pode ser utilizada como auxílio no autoestudo. Como sugestão para um trabalho futuro, poderá ser abordada uma validação mais aprofundada da ferramenta, tendo como personagens os alunos de duas turmas de Algoritmos e Programação. Dessas duas turmas, uma faria uso do sistema Gerador a outra utilizaria os métodos padrões. Ao fim do semestre, poderia ser aplicado um questionário aos alunos a fim de coletar informações sobre suas dúvidas e dificuldades. Além das respostas, suas notas do decorrer da disciplina também poderiam ser utilizadas como métrica. Outra sugestão de trabalho futuro ficaria por conta de um dicionário de termos em uma base de dados externa, que seria acessado pelo sistema Gerador. Assim, diversificaria ainda mais as possibilidades de exercícios gerados, ou também poderia ser integrada com ferramentas de juízes *online* para validar as respostas dos alunos. Estas e outras futuras implementações construirão versões melhoradas da ferramenta, que será uma grande aliada no aprendizado de programação.

REFERÊNCIAS

.NET FIDDLE. **.NET Fiddle**. Disponível em: <https://dotnetfiddle.net>. Acesso em: 07 mai. 2016.

ANTLR. **Another Tool for Language Recognition.** Disponível em: <http://www.antlr.org>. Acesso em: 26 mai. 2016.

ASCENCIO, Ana F. G.; CAMPOS, Edilene A. V. de; **Fundamentos da Programação de Computadores.** São Paulo: Pearson 3ª ed., 2012.

AURELIANO, V. C, O.; TEDESCO, P. C. A. R. Avaliando o uso do Scratch como abordagem Alternativa para o processo de ensino-aprendizagem de programação. XX Workshop sobre Educação em Computação, Curitiba, 2012.

ALBERTI, Taís F.; Mallmann, Elena M.; SONEGO, Anna H. S.; PIGATTO, Giane M.; JACQUES, Juliana S.; STORGATTO, Greyce A. Oportunidades, perspectivas, e limitações dos MOOC no âmbito da UAB/UFSM. **X Congresso Brasileiro de Ensino Superior a Distância,** Belém, 2013. Disponível em: <http://www.aedi.ufpa.br/esud/trabalhos/poster/AT1/114256.pdf>. Acesso em 21 abr. 2016.

CODECADEMY. **Learn to code interactively, for free.** Disponível em: <https://www.codecademy.com>. Acesso em 23 abr. 2016.

DARING FIREBALL. **Markdown**. Disponível em: <https://daringfireball.net/projects/markdown/>. Acesso em 21 mai. 2016.

DETERS, Janice I.; SILVA, Júlia M. C. da; MIRANDA, Elisângela M. de; FERNANDES, Anita M. da R.. O Desafio de Trabalhar com Alunos Repetentes na Disciplina de Algoritmos e Programação. **Simpósio Brasileiro de Informática na Educação**, 2008. Disponível em: <http://www.proativa.virtual.ufc.br/sbie/CD_ROM_COMPLETO/workshops/workshop%202/O%20Desafio%20de%20Trabalhar%20com%20Alunos%20Repetentes%20na.pdf>. Acesso em 16 abr. 2016.

FERRADIN, M.; STEPHANI, S. L. Ferramenta para o ensino de Programação via Internet. In: SULCOMP, 2005. **Anais eletrônicos.** Disponível em: <http://periodicos.unesc.net/index.php/sulcomp/article/viewArticle/794>. Acesso em: 02 mai. 2016.

GIRAFFA, Lucia M. M.; MARCZAK, Sabrina S.; ALMEIDA, Gláucio. O ensino de algoritmos e programação mediado por um ambiente na *web*. WIE, 2003. **Anais eletrônicos.** Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wei/2003/003.pdf>. Acesso em: 02 mai. 2016.

INACIO, Flamarion A. J.; RUFINO, Hugo L. P.; SOUZA, Julio C. de; BALIEIRO, Rogério T.; OLIVEIRA, Paulo H. S. de; MESQUITA, Wisner G.; CASTRO, Paulo A. de. O modelo tradicional de ensino aplicado à disciplina de algoritmos: Estudo de caso e proposta para aperfeiçoamento do método – "É possível desenvolver raciocínio lógico utilizando a teoria das inteligências múltiplas? ". **Enciclopédia Biosfera**, Goiânia, v10, n19, p363, 2014. Disponível em: <https://www.researchgate.net/publication/275153820_MODELO_TRADICIONAL_DE_ENSINO_APLICADO_A_DISCIPLINA_DE_ALGORITMOS_ESTUDO_DE_CASO_E_PROPOSTA_PARA_APERFEICOAMENTO_DO_METODO_-_E_POSSIVEL_DESENVOLVER_RACIOCINIO_LOGICO_UTILIZANDO_A_TEORIA_DAS_INTELIGEN>. Acesso em: 09 abr. 2016.

KLOCK, A. C. T. et al. Análise das técnicas de Gamificação em Ambientes Virtuais de Aprendizagem. **Revista Renote**. CINTED-UFRGS, Porto Alegre, V.12, n.2, dezembro 2014. Disponível em:

<http://www.seer.ufrgs.br/index.php/renote/article/view/53496/33013>. Acesso em: 21 abril 2016.

KNECHTEL, Maria do R. **Metodologia da pesquisa em educação: uma abordagem teórico-prática dialogada.** Curitiba: Intersaberes, 2014.

KURNIA, A; LIM, A.; CHEANG, B. *Online Judge*. ***Computers & Education***, 2001. Disponível em: <https://www.researchgate.net/publication/222285496_Online_Judge>. Acesso em: 21 abr. 2016.

LEARNEROO. ***Learn Programming in Java.*** Disponível em: <https://www.learneroo.com>. Acesso em: 02 mai. 2016.

LIGHTBOT. ***Solve Puzzles using Programming Logic*****.** Disponível em: <https://lightbot.com/>. Acesso em: 18 mai. 2016.

MINGUET, Pilar A. (Org.); **A Construção do Conhecimento na Educação.** Porto Alegre: Artmed, 1998.

PEREIRA, Cláudio de S.; Aprendizagem, educação e trabalho na sociedade do conhecimento. **Revista de Administração Pública,** Rio de Janeiro, v. 35, n. 6, p. 107-117, nov. / dez. 2001.

RADOŠEVIĆ, Danijel; OREHOVAČKI, Tihomir; STAPIĆ, Zlatko. *Automatic On-line Generation of Student's Exercises in Teaching Programming.* ***21st Central European Conference on Information and Intelligent Systems*****,** 2010. Disponível em <http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2505722>. Acesso em 23 abr. 2016.

ROBOCODE. **Build the best - destroy the rest!.** Disponível em: <http://robocode.sourceforge.net/>. Acesso em 23 abr. 2016.

ROCHA, Paulo S.; FERREIRA, Benedito; MONTEIRO, Dionne; NUNES, Danielle da S. C.; GOÉS, Hugo C. do N.. Ensino e Aprendizagem de Programação: Análise da Aplicação de Proposta Metodológica Baseada no Sistema Personalizado de Ensino. **Revista Renote**. CINTED-UFRGS Novas Tecnologias na Educação, v8, n3, 2010. Disponível em: <http://www.seer.ufrgs.br/index.php/renote/article/view/18061/10649>. Acesso em: 16 abr. 2016.

SCRATCH. **Crie histórias, jogos e animações:** partilhe com gente de todo o mundo. Disponível em: <http://scratch.mit.edu>. Acesso em 16 abr. 2016.

SILVA, Willian R. da. **Jogo de tabuleiro: uma ferramenta de auxílio no processo de aprendizagem de algoritmos.** 2015. Trabalho de conclusão de curso.

(Graduação) – Curso de Sistemas de Informação, Centro Universitário UNIVATES, Lajeado, jun. 2016.

SOUZA, C. M. de. VisuAlg – Ferramenta de Apoio ao Ensino de Programação. **Revista TECCEN**. USS, Vassouras, V. 2, n. 2, setembro de 2009. Disponível em <http://www.uss.br/pages/revistas/revistateccen/V2N22009/ArtigoVisuAlgSOUZA.pdf >. Acesso em: 16 abr. 2016.

SPOJ Brasil. ***Online Judge***. Disponível em: <http://br.spoj.com/embed/info>. Acesso em: 15 mai. 2016.

TRAMONTINA, Pedro. **Gerador automático de exercícios para apoio ao ensino de programação.** 2015. Trabalho de conclusão de curso (Graduação) – Curso de Engenharia da Computação, Centro Universitário UNIVATES, Lajeado, nov. 2015.

URI Online Judge: **Problems & Contests**. Disponível em: <https://www.urionlinejudge.com.br>. Acesso em: 15 mai. 2016.

VALENTE, José A.; **Computadores e Conhecimento:** Repensando a Educação. 2 ed. Campinas, SP, 1998.

VIANNA, Ysmar; VIANNA, Maurício; MEDINA, Bruno; TANAKA, Samara. **Gamification, Inc:** Como reinventar empresas a partir de jogos. Rio de Janeiro: MJV Press, 2013, 116p.

YUAN, L; POWELL, S. **MOOCs and Open Education:** Implications for Higher Education. CETIS, 2013. Disponível em: <http://publications.cetis.ac.uk/wp-content/uploads/2013/03/MOOCs-and-Open-Education.pdf>. Acesso em: 21 abril 2016.

APÊNDICES

APENDICE A - PESQUISA DE SATISFAÇÃO

Pesquisa de satisfação do Gerador Automático de Exercícios

Gerador Automático de Exercícios para auxiliar os alunos nas disciplinas de Algoritmos e Programação.

*Obrigatório

1. **1 - Sobre a utilização do sistema Gerador nas disciplinas de Algoritmos e Programação, como você o avalia?** *
   Marcar apenas uma oval.
   - ◯ Extremamente válido
   - ◯ Válido
   - ◯ Pouco válido
   - ◯ Irrelevante

2. **2 - Qual é o grau de complexidade no uso do sistema Gerador? (1 - Pouco complexo e 5 - Muito complexo)** *
   Marcar apenas uma oval.

1	2	3	4	5
◯	◯	◯	◯	◯

3. **3 - Você teve alguma dificuldade no uso do sistema Gerador?** *
   Marcar apenas uma oval.
   - ◯ Sim, muitas dificuldades.
   - ◯ Sim, alguma dificuldade.
   - ◯ Não tive dificuldades.

4. **4 - O formato dos exercícios gerados é didático?** *
   Marcar apenas uma oval.
   - ◯ Sim
   - ◯ Não

5. **5 - Em quais conteúdos você identifica o maior potencial do sistema Gerador, em termos de auxílio aos alunos?** *

Marque todas que se aplicam.

- [ ] Lógica de programação em geral
- [ ] Laços condicionais "if"
- [ ] Laços de repetição "while" e "for"
- [ ] Vetores e Matrizes
- [ ] Procedimentos e Funções
- [ ] Não auxiliará

6. **6 - Em relação a utilização do sistema Gerador, como você classifica a sua experiência?** *

Marque todas que se aplicam.

- [ ] Teve dificuldades em entender como a ferramenta funciona
- [ ] Não teve dificuldades em entender como a ferramenta funciona
- [ ] Teve problemas em entender como os templates funcionam
- [ ] Não tive problemas em entender como os templates funcionam
- [ ] Não conseguiu gerar a lista de exercícios

7. **7 - Qual a sua impressão geral e opiniões em relação ao Gerador Automático de Exercícios?**

.....

.....

.....

.....

.....

APÊNDICE B - *TEMPLATE* SOBRE ARRAYS

```
# Create and print array

### code: arr002
### level: advanced
### topic: array
### tags: array print

## Parameters

<n = fixed integer [3-15]>
<ini = fixed integer [1-10]>
<end = fixed integer [10-20]>

## Description

Create a program that reads two arrays A and B, size ${n}, and
perform the exchange of the elements of these arrays.
That is, after the implementation of the program array B
must contain the values provided for the array A, and vice versa.

## Pseudocode

    declare array vetA of integer size ${n}
    declare array vetB of integer size ${n}
    declare integer ind
    declare integer temp

    for ind from 1 up to ${n} step 1
        vetA[ind-1] = random(${ini}, ${end})
        vetB[ind-1] = random(${ini}, ${end})

    print "Array A: "

    for ind from 1 up to ${n} step 1
        print vetA[ind-1]
        print " "

    println "\n"
    print "Array B: "

    for ind from 1 up to ${n} step 1
        print vetB[ind-1]
        print " "

    println "Changing..."

    for ind from 1 up to ${n} step 1
        temp = vetA[ind-1]
        vetA[ind-1] = vetB[ind-1]
        vetB[ind-1] = temp

    print "Array A: "

    for ind from 1 up to ${n} step 1
        print vetA[ind-1]
        print " "
```

```
    println ""

## Expected results

- Array A = {1,3,5,2} and B = {9,8,7,6} will result A = {9,8,7,6} and B = {1,3,5,2}

## Hints

- Declare two integer arrays and populate with random values.
- Use a repetition, counting from 1 to **"${n}"**.
- Use the operator **+** for concatenating strings.
```

APÊNDICE C - *TEMPLATE* SOBRE MATRIZES

```
# Create and print matrix

### code: mat001
### level: advanced
### topic: matrix
### tags: matrix print

## Parameters

<n = fixed integer [1-10]>

## Description

Write a program that reads a matrix by size ${n}x${n}, calculate
and show the sum of values:
a) line 4;
b) column 2;
c) main diagonal;
d) secondary diagonal;
e) of all elements of the matrix.

## Pseudocode

    declare matrix mat of integer size 5 by 5
    declare integer indLin
    declare integer indCol
    declare integer ind
    declare integer soma
    declare integer somaTotal

    indLin = 0
    indCol = 0
    ind = 0
    soma = 0
    somaTotal = 0

    println "----MATRIZ----"
    while indLin < 5
        indCol = 0
        while indCol < 5
            mat[indLin][indCol] = random(1,100)
            print mat[indLin][indCol] + " "
            somaTotal = somaTotal + mat[indLin][indCol]
            indCol = indCol + 1
        println "\n"
        indLin = indLin + 1

    println "-----------------"

    // Item A
    soma = 0
    indCol = 0
    for indCol from 0 up to 4 step 1
        soma = soma + mat[3][indCol]

    println "A soma dos elementos da linha 4 é: " + soma
```

```
    // Item B
    soma = 0
    indLin = 0
    for indLin from 0 up to 4 step 1
        soma = soma + mat[indLin][1]

    println "A soma dos elementos da coluna 2 é: " + soma

    // Item C
    soma = 0
    ind = 0
    for ind from 0 up to length(mat) - 1 step 1
        soma = soma + mat[ind][ind]

    println "A soma dos elementos da diagonal principal é: " + soma

    // Item D
    soma = 0
    ind = 0
    for ind from 1 up to length(mat) - 1 step 1
        soma = soma + mat[ind][(length(mat) - 1) - ind]

    println "A soma dos elementos da diagonal secundária é: " + soma

    //Item E
    println "A soma de todos elementos da matriz é: " + somaTotal

## Expected results


## Hints

- Declare an integer matrix and populate with random values.
- Use a repetition, counting from 1 to **"${n}"**.
- Use the operator **+** for concatenating strings.
```

APÊNDICE D - *TEMPLATE* SOBRE PROCEDIMENTOS

```
# Print multiplication table

### code: pro001
### level: advanced
### topic: procedure
### tags: procedure print

## Parameters

<n = fixed integer [1-10]>

## Description

Create a static method (procedure) which receives a number by
parameter and writes the multiplication table 10 to that number.

## Pseudocode
    procedure tabuada (integer n)
        declare integer cont
        cont = 1
        println "Tabuada do número " + n
        while cont <= 10
            println n + " x " + cont + " = " + n * cont
            cont = cont + 1

    declare integer num
    read num
    tabuada(num)

## Expected results



## Hints

- Create a procedure which receives a number by parameter
- Use a repetition, counting from 0 to 10.
- Use the operator **+** for concatenating strings.
```

APÊNDICE E - *TEMPLATE* SOBRE FUNÇÕES

```
# Category of swimmer

### code: fun002
### level: advanced
### topic: function
### tags: function

## Parameters

<n = fixed integer [1-10]>

## Description

Create a static method (Function) that receives the age of the swimmer by
parameter and returns the category of him, according the table below:
    5  to  7 years old                    - INFANT A
    8  to 10 years old                    - INFANT B
    11 to 13 years old                    - JUVENILE A
    14 to 17 years old                    - JUVENILE B
    Higher than 18 years old (including) - ADULT

## Pseudocode
    function string categoria ( integer num )
        if num <= 7
            return "INFANT A"
        else
            if num <= 10
                return "INFANT B"
            else
                if num <= 13
                    return "JUVENILE A"
                else
                    if num <= 17
                        return "JUVENILE B"
                    else
                        return "ADULT"

    declare string cat
    declare integer idade
    read idade

    while idade < 5 or idade > 120
        println "INVALID age! Type a value between 5 and 120."
        read idade

    cat = categoria ( idade )
    println "The swimmer is of category " + cat

## Expected results


## Hints

- Use a repetition, counting from 1 to **"${n}"**.
- Use the operator **+** for concatenating strings.
```

APÊNDICE F - MANUAL DE INSTRUÇÕES

MANUAL DO GERADOR AUTOMÁTICO DE EXERCÍCIOS

Este manual tem por objetivo esclarecer dúvidas quanto ao processo de geração e visualização dos exercícios.

Conteúdo da pasta

Link para a pasta:
https://drive.google.com/open?id=0B7Emisap6BdzTENGWFhsT0hwS3c

Generated_Exercises
Este é o diretório onde serão gravados os arquivos dos exercícios gerados.

lib
Diretório que contém bibliotecas (ANTLR e FreeMarker) que são utilizadas pelo Generator.jar.

Templates
Diretório onde estão armazenados os templates que servem de base para os exercícios gerados.

generate.bat
Arquivo executável criado para automatizar a geração dos exercícios, assim, não é preciso digitar linha de comando no prompt para iniciar o processo de geração.

Generator.jar
Projeto do Gerador Automático de Exercícios.

Passo a passo da geração dos exercícios

Este projeto já conta com alguns templates prontos que servem de base na geração dos exercícios. Eles se encontram na pasta Templates. Os tópicos abordados nos templates são arrays, matrizes, procedimentos e funções.

Para disparar o processo de geração dos exercícios, o arquivo Generator.jar deverá ser chamado via linha de comando. Para otimizar este processo, foi criado o arquivo generator.bat. Em seus comandos, há uma chamada ao Generator.jar e alguns parâmetros são passados a ele:

-t exerciselist -lv advanced -i array -n 5 -ln java -h advanced -o Generated_Exercises

-t
Este parâmetro poderá ter somente 2 valores: exerciselist ou testlist. O parâmetro exerciselist significa que quando informado, será gerada uma lista de exercícios. O parâmetro testlist significa que quando informado, será gerada uma lista de exames, onde cada arquivo gerado tem vários enunciados de exercícios e as respostas são geradas em um arquivo separado. Quando o parâmetro -t for testlist, no parâmetro -i terá de ser informado o número dos exercícios, por exemplo, -i 1 array 2 function 3 procedure. Isso significa que o primeiro exercício do teste será sobre array, o segundo sobre funções e o terceiro sobre procedimentos.

-lv
Este parâmetro indica o nível de dificuldade dos exercícios. Pode assumir os valores simple, medium e advanced. Os templates existentes são somente do nível advanced.

-i
Este parâmetro trata dos tópicos abordados nos exercícios, por exemplo. -i string input math. Isso significa que serão selecionados os templates que sejam do tópico string e possuem tags input ou math.

-n
Este parâmetro indica o número de exercícios (arquivos) que serão gerados.

-ln
Parâmetro que indica a linguagem de saída do código presente no exercício. Pode assumir os valores C, C++, Java, Python e Portuguese. Para simplificar, o script gerará exercícios na liguagem Java.

-h
Parâmetro que indica o nível das dicas geradas nos exercícios. Pode ter os valores basic, medium e advanced.

-o
Parâmetro que indica o diretório onde serão salvos os exercícios gerados. Neste exemplo, o parâmetro -o recebe o valor Generated_Exercises.

Visualização dos exercícios gerados

Após o arquivo generate.bat ser executado, os exercícios serão gerados e salvos no diretório Generated_Exercises. Esses arquivos possuem a extensão .md, pois são arquivos gerados na linguagem de marcação Markdown. Para visualizar os exercícios formatados, deverá ser instalado o software MarkdownPad.
Se por acaso ocorrerem erros na renderização, ou visualização dos exercícios, deverá ser instalado o AwesomiumSDK, pois o mesmo possui algumas bibliotecas que o MarkdownPad utiliza na renderização os arquivos .md.