UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ
CAMPUS LUIZ MENEGHEL - CENTRO DE CIÊNCIAS TECNOLÓGICAS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

VINÍCIUS OVILE SAQUES

ANÁLISE DA FERRAMENTA ROBOCODE PARA APRENDIZADO DE ORIENTAÇÃO A OBJETOS

BANDEIRANTES–PR

2018

VINÍCIUS OVILE SAQUES

ANÁLISE DA FERRAMENTA ROBOCODE PARA APRENDIZADO DE ORIENTAÇÃO A OBJETOS

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Sistemas de Informação da Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

BANCA EXAMINADORA

Prof. Me. José Reinaldo Merlin
Universidade Estadual do Norte do Paraná
Orientador

Prof. Dra. Daniela de Freitas Guilhermino Trindade
Universidade Estadual do Norte do Paraná

Prof. Me. Fabio de Sordi Junior
Universidade Estadual do Norte do Paraná

Bandeirantes–PR, 01 de novembro de 2018

VINÍCIUS OVILE SAQUES

ANÁLISE DA FERRAMENTA ROBOCODE PARA APRENDIZADO DE ORIENTAÇÃO A OBJETOS

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Sistemas de Informação da Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. José Reinaldo Merlin

BANDEIRANTES–PR

2018

OVILE, V. **Análise da Ferramenta Robocode para Aprendizado de Orientação a Objetos**. 35 p. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Universidade Estadual do Norte do Paraná, Bandeirantes–PR, 2018.

RESUMO

As disciplinas de programação são básicas em todos os cursos da área de computação. No entanto, estas disciplinas tradicionalmente possuem alto índice de reprovação e contribuem para a evasão nestes cursos. Para ajudar a minimizar o problema, diversas ferramentas tem sido propostas. Neste trabalho foi analisada a ferramenta Robocode, que aborda a programação orientada a objetos. Foram pesquisados quais os conteúdos ministrados nas disciplinas de programação que abordam este paradigma. Estes conteúdos foram confrontados com as funcionalidades de Robocode, afim de analisar se a ferramenta cobre os temas ministrados. Constatou–se que a ferramenta pode ser útil ao aprendizado de orientação a objetos de forma parcial, pois são necessários conhecimentos prévios para poder utilizá–la.

Palavras-chave: Aprendizagem de Programação. Orientação a Objetos. Robocode.

OVILE, V.. **Robocode Tool Analysis for Object Oriented Learning**. 35 p. Final Project (Bachelor of Information Systems) – State University Northern of Parana , Bandeirantes–PR, 2018.

ABSTRACT

Programming courses are basic in all courses in the area of computing. However, these disciplines traditionally have a high rate of disapproval and contribute to avoidance in these courses. To help minimize the problem, several tools have been proposed. In this work the Robocode tool, which approaches object oriented programming, was analyzed. It was researched which contents were taught in the programming disciplines that approach this paradigm. These contents were confronted with the functionalities of Robocode, in order to analyze if the tool covers the themes taught. It was verified that the tool can be useful to the learning of object orientation in a partial way, since previous knowledge is necessary to be able to use it.

Keywords: Programming Learning. Object Orientation. Robocode.

SUMÁRIO

1 INTRODUÇÃO

Os cursos de Graduação em Computação, segundo o MEC, se dividem em: Ciência da Computação, Sistemas de Informação, Engenharia da Computação e Licenciatura em Computação[15], embora existam cursos com outras denominações. Nestes cursos, as disciplinas de programação são básicas. Ao mesmo tempo, estas mesmas disciplinas são as que apresentam maior índice de reprovação, contribuindo para alta evasão nestes cursos [12]. Fernandes e Junior [5] explicam que esses índices são elevados porque os alunos apresentam dificuldades para entender a lógica de programação, que requer uma prática intensiva para ser desenvolvida. Raabe e Silva [13] dizem que essas dificuldades podem se dar, também, pelo alto nível de abstração do conteúdo.

Esta situação tem motivado vários pesquisadores a buscarem soluções criativas para o ensino de programação, tais como a utilização de jogos e o emprego de gamificação. Neste trabalho é investigada uma terceira vertente, que é a utilização de ferramentas para ensino de programação, mais precisamente o ambiente Robocode para ensino de programação orientada a objetos (POO). Esta ferramenta foi escolhida porque, diferentemente das demais, o usuário tem a necessidade de escrever código, na linguagem Java ou plataforma .NET. Desta forma, é importante verificar o quanto pode ser útil no ensino de POO.

1.1 Objetivos

O objetivo deste trabalho é realizar uma análise da cobertura da ferramenta Robocode em relação aos conteúdos ministrados nas disciplinas de programação orientada a objetos.

O presente trabalho tem como Objetivos Específicos:

- Elencar os conteúdos abordados nas disciplinas de Programação Orientação a Objetos;
- Confrontar os temas de orientação a objetos (OO) com as funcionalidades do Robocode; e
- Analisar o quanto Robocode cobre dos conteúdos estudados.

1.2 Justificativa

Enfrentar o alto índice de reprovação nas disciplinas de programação tem sido um desafio para os professores da área [12]. Existem diversas ferramentas e abordagens para auxiliar o aprendizado de programação, tais como Scratch, Alice, AppInventor e ambientes

virtuais de aprendizagem. No entanto, para boa parte destas, não existe correlação entre suas funcionalidades e os conteúdos ministrados nas disciplinas de programação. Como exemplo, pode–se citar o tema "ponteiros", não abordado por nenhuma das ferramentas pesquisadas.

Por isso, faz-se importante conhecer e analisar as diversas ferramentas existentes confrontando–as com os conteúdos ministrados, para avaliar o quanto elas cobrem destes conteúdos.

1.3 Abordagem Metodológica

O presente trabalho se caracteriza, quanto ao objetivo, como pesquisa exploratória, pois visa explorar ideias [16] e proporcionar maior familiaridade com o tema. Não foram encontrados na literatura estudos sobre a cobertura de conteúdos na forma como é abordada neste trabalho.

As etapas de desenvolvimento deste trabalho foram:

- Revisão de literatura sobre o tema;
- Levantamento dos conteúdos ensinados nas disciplinas de programação orientada a objetos; e
- Estudo sobre como os temas comuns em orientação a objetos podem ser abordados utilizando-se Robocode.

1.4 Organização do Trabalho

Neste capítulo o trabalho foi contextualizado, foram apresentados os objetivos, a justificativa e método de desenvolvimento. No capítulo seguinte é apresentada fundamentação teórica sobre aprendizagem de programação e ferramentas existentes. No Capítulo 3, os conteúdos ministrados nas disciplinas são comparados com as funcionalidades do Robocode. Ao final, no Capítulo 4, são apresentadas as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos que embasam o desenvolvimento do trabalho. Inicialmente, são apresentados alguns estudos sobre as dificuldades no aprendizado de programação e, posteriormente, algumas ferramentas para auxílio a este aprendizado.

2.1 Ensino e Aprendizagem de Programação

Nos cursos da área de computação, as disciplinas de programação são básicas. Estas disciplinas estão entre as que apresentam maior índice de reprovação, contribuindo para alta evasão nestes cursos [12].

Gomes *et al.* [7], analisando as causas da alta taxa de reprovação, dizem que os métodos de ensino não são adequados para muitos estudantes, pelo fato de o ensino não ser personalizado e cada aluno aprender de uma forma diferente. Outro motivo de inadequação é que, muitas vezes, o foco é a linguagem de programação ao invés de promover a capacitação para resolução de problemas.

Para Viegas *et al.* [17], antes da construção do programa, o estudante precisa visualizar o que deve ser feito para desenvolver o código fonte, o que muitas vezes é difícil para a maioria deles. Isto corrobora com Raabe e Silva[13], que dizem que as dificuldades podem se dar pelo alto nível de abstração exigido para o desenvolvimento de programas.

A estratégia mais comum para apresentar conteúdos em disciplinas é a utilização de aulas expositivas. Para Choi e Hannafin [4], este tipo de estratégia instrucional é adequada para apresentar conceitos abstratos e informações factuais para um grande grupo de alunos. No entanto, quando se pretende atingir objetivos de aprendizagem em níveis mais aprofundados, que promovam a aplicação do conhecimento em situações práticas, não é o mais adequado. Sendo assim, para o autor citado, o ideal é que se utilizem ambientes de aprendizagem.

Para Oliveira [10], um ambiente de aprendizagem pode ser entendido como um espaço das relações com o saber, este sendo o objeto maior do processo de aprendizagem. Para os autores, estes espaços devem ser vistos como ambientes que favorecem a construção do conhecimento que ocorre a partir das interações dos alunos com os conteúdos, com os outros alunos e com os professores.

No entanto, apesar de todos os estudos evidenciando a importância das ferramentas, pesquisa feita por Oliveira[11] mostrou que, durante a graduação, 60% dos estudantes não utilizaram nenhum tipo de ferramenta ou ambiente de apoio. Ainda segundo Oliveira[11], dos 40% que utilizaram alguma ferramenta, 32% utilizaram laboratório de

informática, 8% usaram ferramentas como *Scratch*, 3% valeram-se de plataformas de cursos online, 10% fizeram uso do ambiente *Moodle* e 8% utilizaram sistemas de envio e correção como o *Run.codes*.

2.2 Jogos e ensino de computação

A aprendizagem baseada em jogos (*Game-based Learning - GBL*) tem sido empregada objetivando tornar os conteúdos mais atrativos aos alunos. A GBL trata de aplicações de jogos nas quais o resultado é a aprendizagem de um determinado conteúdo. Geralmente, estes jogos combinam o conteúdo com a jogabilidade [3].

Neste sentido, um jogo pode ser definido como "qualquer competição (jogo) entre os adversários (jogadores) que operem sob restrições (regras) para um objetivo (vitória ou lucro)" [3]. Ainda segundo Battistella *et al.*, jogos educacionais, também chamados de jogos sérios, são especificamente projetados para ensinar as pessoas acerca de um determinado assunto, expandir conceitos, reforçar o desenvolvimento, exercitando ou aprendendo uma habilidade [3].

A utilização de jogos torna o aprendizado mais divertido e atrativo. Battistella *et al.* identificam diversos jogos para o ensino de computação:

- *Problems and Programers* e *X-MED*: para ensino de Engenharia de Software;
- *No Bug's Snack Bar*, *C-Jump*, *Saving Princess Sera*: para ensino de programação; e
- *Anti-Phishin Phil*: para área de segurança.

O jogo provoca o envolvimento do aluno durante o aprendizado e este envolvimento tem papel chave na qualidade e profundidade do aprendizado [3].

2.3 Ferramentas de Apoio à Aprendizagem de Programação

Rosa e Giraffa[14] ressaltam a importância da existência de ferramentas que busquem despertar interesse nos estudantes e facilitar o entendimento de lógica de programação, pois este entendimento é um desafio enfrentado por muitos deles. Nas próximas subseções são apresentadas algumas delas.

2.3.1 Alice

Alice é um ambiente de programação baseado em blocos que permite a criação de animações, narrativas interativas e pequenos jogos por meio de um ambiente gráfico interativo,

onde o usuário arrasta as instruções para criar o programa. O diferencial é que este ambiente nos permite criar o conteúdo em 3D, e tem o foco de ensinar a lógica de programação orientada a objetos[6]. Na Figura 1 é apresentada uma tela do ambiente Alice.

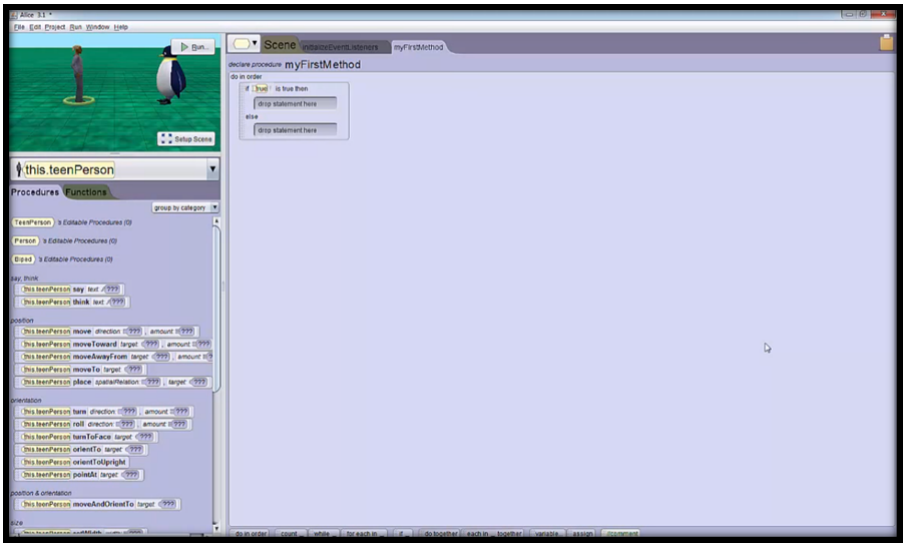

Figura 1 – Tela do ambiente Alice.

2.3.2 App Inventor

App Inventor é um ambiente de programação visual baseado em blocos desenvolvido pela empresa Google (Figura 2). Este ambiente foi criado com o intuito de fazer com que qualquer pessoa, tenha ela noção de informática ou não, seja ela adulta, adolescente ou criança, crie um aplicativo para o seu celular. No entanto, este ambiente só cria aplicativos para os celulares Android, que é o sistema operacional da empresa Google[6].

Figura 2 – Tela do ambiente App Inventor.

2.3.3 Scratch

Scratch é uma linguagem de programação visual que tem o propósito de introduzir a lógica de programação estruturada e conceitos matemáticos. Pode-se criar histórias, animações, jogos, entre outras coisas. Não é necessário digitar nenhum texto ou criar nenhuma função, basta agrupar os comandos que já estão prontos[6], arrastando-se os blocos mostrados na Figura 3. Com esta ferramenta os estudantes conseguem desenvolver raciocínio lógico, capacidade de abstração e criatividade mais facilmente [17].

Figura 3 – Tela do ambiente Scratch.

2.3.4 Robocode

Robocode foi desenvolvido por AlphaWorks, divisão da IBM, com o objetivo de divulgar novas tecnologias de desenvolvimento[2]. Trata-se de um ambiente de simulação de batalha de robôs, em que se programa em Java. O objetivo principal de um jogador é utilizar as classes bases presentes no ambiente na criação do seu próprio robô e colocá-lo em batalha com outros robôs criados por outros desenvolvedores. Para programar um robô, cria-se uma classe derivada das classes existentes no Robocode. Pode-se criar robôs guerreiros dos mais variados tipos. Os robôs possuem radares, podem atirar, colidem entre si e movimentam-se livremente. O ambiente contém um editor de código-fonte próprio para programar os robôs e um espaço para criar uma arena e configurar a batalha[2]. Na Figura 4 são mostrados dois robôs em uma ambiente típico de batalha.

A programação basicamente se dá por meio da definição do comportamento dos robôs nos métodos das classes. Estes métodos definem as ações dos robôs. O tratamento de eventos também é crucial no desenvolvimento dos objetos. Existem, no ambiente, eventos para avisar o robô quando ele acerta ou recebe um tiro, quando ele é eliminado da batalha ou quando ela é finalizada.

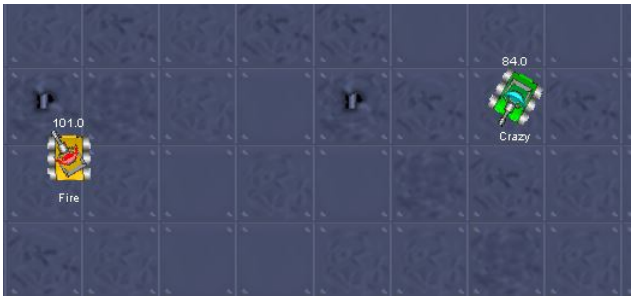

Figura 4 – Tela do ambiente Robocode.

Mais precisamente no que se refere ao aprendizado de programação, o ambiente tem o intuito de desenvolver a lógica de programação e o aprendizado dos conceitos de orientação a objetos[6]. Também é possível introduzir inteligência artificial nos robôs, mas este conceito não será abordado neste trabalho. Na Figura 5, extraída de [2], são mostradas as possibilidades de movimento que podem ser programadas pelo usuário. Pode-se alterar a direção de rotação do robô e do canhão, que podem ser movimentadas para a direita ou esquerda, individualmente ou em conjunto; pode-se, também, alterar o poder de fogo, a direção e velocidade da bala, entre outras coisas.

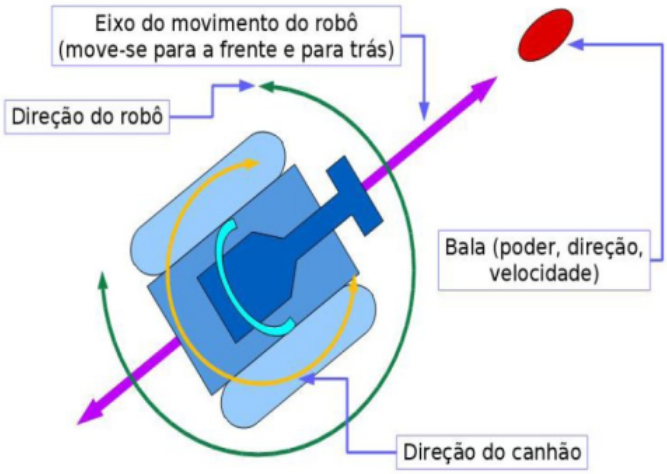

Figura 5 – Características relacionadas ao movimento do robô.

Para Amaral *et al.*[2], por meio de Robocode os estudantes podem aprender com um jogo, que é um método que vem cada vez mais se mostrando promissor.

2.4 Considerações finais

Neste capítulo foram expostas algumas das dificuldades enfrentadas na aprendizagem de programação. Também foram mostradas algumas ferramentas para apoiar o aprendizado.

No próximo capítulo o ambiente Robocode será analisado mais detalhadamente.

3 ORIENTAÇÃO A OBJETOS NO ROBOCODE

Há dois paradigmas principais que são trabalhados nas disciplinas introdutórias de programação: procedimental (muitas vezes chamado de estuturado) e orientado a objetos. Geralmente, nas séries iniciais, são trabalhados conceitos de lógica de programação utilizando-se o paradigma procedimental e, nas séries seguintes, conceitos de orientação a objetos.

Como o foco deste trabalho é a programação orientada a objetos, para o desenvolvimento dele, inicialmente foram pesquisados quais os conteúdos são trabalhados nas disciplinas de programação cujo conteúdo é o paradigma OO. Destes conteúdos, foram selecionados os que estão presentes na maioria dos cursos e estes foram analisados com mais profundidade.

3.1 Conteúdos abordados em Orientação a Objetos

Para o levantamento dos conteúdos trabalhados nas disciplinas cujo foco é orientação a objetos, foram selecionadas 10 instituições de ensino de renome. Os cursos oferecidos nestas instituições são Ciência da Computação, Sistemas de Informação e Análise e Desenvolvimento de Sistemas.

Os temas vistos podem ser divididos em dois grandes grupos: conceitos abstratos e especificidades da linguagem. Como conceitos abstratos entende-se aqueles que são independentes da linguagem utilizada na disciplina. Por exemplo, o conceito de classe e objeto é comum a qualquer linguagem, seja ela Java ou Python. Por outro lado, alguns assuntos são específicos da linguagem, como por exemplo, o conceito de *garbage collector*, presente na linguagem Java mas que não existe em C++. No presente trabalho deu-se mais importância aos conceitos comuns, embora a ferramenta (Robocode) utilize a linguagem Java.

Ainda sobre os temas, é importante salientar que os documentos que contém os conteúdos programáticos não são uniformes. Percebe-se que alguns são mais genéricos enquanto outros apresentam-se mais específicos. Assim, embora o conceito de "atributo" só apareça em três dos cursos analisados, é praticamente impossível que não seja abordado na disciplina. Supõe-se que este conceito esteja incluído no tópico "Classes e objetos", mesmo não sendo mencionado explicitamente.

Nos cursos analisados, foram identificados 24 temas abordados nas disciplinas, mostrados na Tabela 1.

As instituições e cursos analisados foram os mostrados na Tabela 2.

Tabela 1 – Conteúdos de Orientação a Objetos.

Assunto/Instituição	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Classes e Objetos	X	X	X	X	X	X	X	X	X	X
Agregação							X	X	X	X
Herança	X	X	X	X	X	X	X	X	X	X
Generalização							X		X	X
Polimorfismo	X	X	X	X	X	X	X	X	X	X
Encapsulamento	X		X	X		X	X	X	X	X
Modificadores	X						X			
Exceções	X	X		X	X				X	X
Programação Genérica	X			X						X
Interfaces	X	X			X	X			X	X
Estruturas Fundamentais			X	X						
Características Gerais		X	X	X	X	X		X		X
Composição		X							X	
Pacotes e Classes Utilitárias		X					X			
Streams	X									
Serialização	X									
Persistência em Arquivos		X							X	
Atributos			X			X				X
Métodos			X		X	X			X	X
Sobrecarga			X		X	X			X	X
Sobrescrita				X						
Conversão de Tipos						X				
Módulos							X			
Frameworks Orientados a Objetos							X			X

Tabela 2 – Instituições e cursos analisados.

	Instituição	**Curso**
1	UENP - Universidade Estadual do Norte do Parná	Ciência da Computação
2	IFES - Instituto Federal do Espírito Santo	Análise e Desenvolvimento de Sistemas
3	UTFPR - Universidade Tecnológica Federal do Paraná	Sistemas de Informação
4	USP - Universidade de São Paulo	Ciência da Computação
5	IFC - Instituto Federal Catarinense	Sistemas de Informação
6	IFRN - Instituto Federal do Rio Grande do Norte	Sistemas de Informação
7	UFPA - Universidade Federal do Pará	Ciência da Computação
8	UFRS - Universidade Federal do Rio Grande do Sul	Ciência da Computação
9	Anhanguera Educacional	Ciência da Computação
10	UDESC - Universidade do Estado de Santa Catarina	Ciência da Computação

Pode–se verificar, observando os dados apresentados na Tabela 1, que alguns conteúdos aparecem em todas as instituições, enquanto outros são específicos a algumas delas.

3.2 Conteúdos analisados com relação a Robocode

Dos 24 conceitos elencados na Tabela 1, foram selecionados 19, considerados os mais relevantes para o autor desta pesquisa. Estes conteúdos selecionados foram analisados para verificar se são cobertos por Robocode e, em caso positivo, como são aplicados ao se programar os robôs.

3.2.1 Classes e Objetos

O primeiro conteúdo elencado é fundamental para a Programação Orientada a Objetos (POO), visto que um sistema OO é formado a partir de um conjunto de objetos inter-relacionados que possuem propriedades e operações próprias. É fato que essas características (propriedades e operações) são descritas e armazenadas por meio de classes que representam cada uma das diversas entidades que integram um sistema OO [9].

Mais especificamente, pode-se definir classes como um modelo para a construção dos objetos. As classes definem as especificações (estados, comportamento e identidade) dos objetos. Por sua vez, os objetos são definidos por instâncias de classes, isto é, entidades reais construídas em memória [8, 9].

Esses dois conceitos estão presentes nos algoritmos desenvolvidos para os robôs no Robocode, pois todo robô é construído em uma classe. O algoritmo mostrado a seguir indica uma classe padrão para criação de um robô:

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends Robot {
3      // comportamento padrao do robo
4    public void run() {
5      while(true) {
6        ahead(100);
7        turnGunRight(360);
8        back(100);
9        turnGunRight(360);
10     }
11   }
12     // comportamento ao ver outro robo
13   public void onScannedRobot(ScannedRobotEvent e) {
14     fire(1);
15   }
```

```
16      // comportamento ao ser atingido
17    public void onHitByBullet(HitByBulletEvent e) {
18      back(10);
19    }
20    // comportamento ao colidir com a parede
21    public void onHitWall(HitWallEvent e) {
22      back(20);
23    }
24  }
```

Listagem 3.1 – Exemplo de classes no Robocode.

Quanto ao conceito de objetos, pode-se afirmar que sempre quando um robô do tipo *MeuRobo* for criado (instanciado), será criado um novo objeto em memória que poderá executar os comportamentos descritos pela classe.

3.2.2 Agregação e Composição

O conceito de Agregação pode ser entendido como uma forma de relacionamento entre classes. Uma agregação ocorre quando há uma associação do tipo todo/parte, em outras palavras, quando os objetos de uma determinada classe fazem parte (estão contidos) de outro objeto. É importante ressaltar que, nesse relacionamento, o objeto que é composto pelos outros objetos não é responsável pela criação e destruição dos objetos agregados. Comumente, emprega-se uma lista de objetos agregados. [9]

A composição é uma abstração semelhante à agregação. A diferença fundamental entre elas é o fato de a composição pressupõe uma relação de dependência entre os objetos, pois os objetos agregados passam a pertencer a outro objeto, o qual é responsável por controlar, criar e destruir seus agregados.

Ambos os conceitos apresentados nessa seção não foram considerados diretamente aplicáveis no ambiente Robocode.

3.2.3 Herança e Generalização

Este tópico também é muito importante para a utilização do Robocode. Os conceitos de Herança e Generalização são intimamente relacionados, porém não possuem o mesmo significado. A generalização é um dos tipos de relacionamento possíveis entre classes, na qual uma determinada classe possui versões refinadas (especializadas, subclasses ou classes derivadas) de si mesma (superclasse ou classe base). Dessa forma, esse relacionamento é do tipo “é um”, pois uma instância da subclasse também é uma instância da superclasse [9].

Cada subclasse recebe as características (atributos e comportamentos) da sua superclasse, mas é possível adicionar-se novas características, conforme necessário. Essa ação de receber características de outra classe é chamada de Herança, pois a subclasse herda de sua superclasse. Em Java, emprega-se a palavra reservada *extends* para indicar qual é a superclasse da qual herda-se características.

Dentro do ambiente Robocode, essas abstrações são fundamentais, visto que não se constrói um robô inteiramente novo. Para a criação dos robô, é necessário implementar uma especialização da classe *Robot* ou *JuniorRobot*. O exemplo a seguir indica uma parte de uma classe especializada de *Robot* que possui características próprias como, por exemplo, as cores para o robô e seu comportamento padrão.

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends Robot {
3
4      // comportamento padrao do robo
5    public void run() {
6      setColors(orange, blue, white, yellow, black);
7      while(true) {
8        ahead(100);
9        turnGunRight(360);
10       back(100);
11       turnGunRight(360);
12     }
13   }
14 }
```

Listagem 3.2 – Exemplo de Herança e Generalização no Robocode.

3.2.4 Polimorfismo

O Polimorfismo em Java relaciona-se ao processo de referência de objetos, qual seja, associar uma determinada variável a um determinado objeto existente na memória. Quando uma generalização é aplicada entre classes, é possível a criação de uma variável de referência polimórfica, a qual espera-se uma referência à sua superclasse, porém é possível a atribuição de objetos da própria subclasse. Isso ocorre devido ao fato de um objeto da subclasse ser uma instância da superclasse também [8].

Embora o Polimorfismo relaciona-se diretamente com a abstração de Generalização, não se percebe uma aplicação clara, em linhas de código, do mesmo na construção de robôs no ambiente Robocode. Uma possível aplicação de polimorfismo seria uma lista de elementos do tipo *Robot* populada com elementos que sejam subclasses daquela. Poder-se-ia percorrer a lista invocando métodos da superclasse, sendo que, em tempo de execução,

seriam executados o método correspondente ao elemento real da lista.

3.2.5 Encapsulamento e Modificadores

Encapsulamento é um conceito importante na POO, uma vez que o mesmo consiste na aplicação de técnicas de proteção nos atributos de uma determinada classe para que esses não possam ser alterados por outras classes. Assim sendo, define-se um conjunto de métodos na classe que contém os atributos encapsulados para que as demais classes os acionem. Tais métodos são os responsáveis por fazer as alterações necessárias nos atributos de sua classe [9].

Quando se realiza o encapsulamento de atributos, comumente são gerados métodos do tipo *Getter*, *Setter*, *Checker* e *Converter*:

- ***Getter***: método utilizado para retornar o valor de um determinado atributo;
- ***Setter***: método utilizado para definir o valor de um determinado atributo;
- ***Checker***: método utilizado para verificar uma característica de um determinado atributo, retornando verdadeiro ou falso; e
- ***Converter***: método utilizado para converter o valor de um determinado atributo para um outro tipo.

O encapsulamento é uma abstração intimamente relacionada aos modificadores de acesso. Tais modificadores são responsáveis por indicar quais outras classes poderão acessar os atributos e métodos de uma determinada classe. Em Java existem três modificadores de acesso principais:

- ***public***: indica que qualquer outra classe poderá fazer uso do atributo ou método em questão;
- ***private***: indica que apenas a classe em questão poderá acessar um determinado atributo ou método; e
- ***protected***: indica que apenas a própria classe, suas possíveis subclasses e as demais classes em seu pacote poderão acessar um determinado atributo ou método.

Segundo [8], além dos modificadores de acesso, é possível elencar mais dois modificadores relevantes para a POO:

- ***static***: indica que um determinado atributo ou método pertence a uma classe e não aos seus objetos. Dessa forma, tal elemento será compartilhado e único para todas as instâncias dessa classe; e

- ***final***: indica que o valor de determinado atributo não é mutável, em outras palavras, uma vez definido não poderá ser alterado.

Esses conceitos são facilmente perceptíveis no ambiente Robocode. Observando-se o fragmento de código é possível notar o uso do modificador de acesso *public* tanto para a classe MeuRobo quanto para o método *run*. Além disso, emprega-se a chamada de dois métodos dos tipos *Getter* e *Setter* para retornar o valor atual da energia do robô e alterar as cores do robô, respectivamente. Isso é necessário por os atributos de energia e cores serem encapsulados na superclasse *Robot*.

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends Robot
3  {
4
5      // comportamento padrao do robo
6    public void run() {
7        double energia = getEnergy();
8        setColors(orange, blue, white, yellow, black);
9      // ... continuacao
10   }
11 }
```

Listagem 3.3 – Exemplo de Encapsulamento e Modificadores no Robocode.

3.2.6 Exceções

No contexto da programação, exceções são situações em que uma determinada ação ou evento não ocorrer como o esperado. É válido que as exceções podem ter origem a partir de diversas causas como, por exemplo, entradas ou valores incorretos ou incompatíveis, erros em dispositivos, limitação de recursos físicos ou, até mesmo, erros no próprio código. Em linguagens OO, como Java, é possível realizar a manipulação dessas exceções, auxiliando na melhora do programa gerado por permitir uma maior "prevenção" para os possíveis erros em tempo de execução desse programa [8].

No contexto da programação em Java, quando há uma exceção, é comum retornar-se um objeto que encapsule o erro ocorrido e suas informações para que se possa realizar alguma correção ou tratamento do mesmo. Para isso, comumente emprega-se a cláusula *throws* (literalmente uma cláusula de retorno para uma exceção) ou uma estrutura *try-catch* (tenta-se realizar uma determinada ação e, se a mesma falhar, o código de tratamento é invocado).

Tratamentos de exceções não foram encontrados com muita frequência nos robôs do RoboCode, porém no fragmento de código a seguir as exceções são manipuladas para

garantir a correta manipulação de um arquivo de entrada para contagem da participação do robô em outras batalhas. A manipulação é feita através da estrutura *try-catch* com o tratamento de exceções de entrada/saída (*IOException*) e conversões numéricas (*NumberFormatException*).

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends AdvancedRobot
3  {
4
5      // comportamento padrao do robo
6    public void run() {
7      int roundCount, battleCount;
8      try {
9        BufferedReader reader = null;
10       try {
11         reader = new BufferedReader(new FileReader(getDataFile("
               count.dat")));
12
13         roundCount = Integer.parseInt(reader.readLine());
14         battleCount = Integer.parseInt(reader.readLine());
15
16       } finally {
17         if (reader != null) {
18           reader.close();
19         }
20       }
21     } catch (IOException e) {
22       roundCount = 0;
23       battleCount = 0;
24     } catch (NumberFormatException e) {
25       roundCount = 0;
26       battleCount = 0;
27     }
28     // ... continuacao
29   }
30   // ... continuacao
31 }
```

Listagem 3.4 – Exemplo de tratamento de Execoes no Robocode.

3.2.7 Programação Genérica

A abstração de Programação Genérica pode ser considerada uma ferramenta útil para economia e otimização de código. Esse conceito não é tão facilmente encontrado sendo diretamente aplicado em pequenos sistemas, porém é importante ressaltar que é empregada internamente em linguagens de programação como o Java, por exemplo. Esse tipo de programação é responsável por evitar a rescrita de uma classe ou método para adequá-los a objetos específicos como, por exemplo, apenas receber objetos do tipo *String* ou *Integer*. Por meio da Programação genérica, é possível substituir o tipo do objeto por uma indicação genérica *T*. Essa marca genérica é muito utilizada na construção de listas e coleções de objetos em Java como, por exemplo, uma *ArrayList*, que é genérica [8].

Essa abstração não é aplicável diretamente na construção de robôs no ambiente RoboCode.

3.2.8 Interfaces

As *Interfaces* são comumente aplicadas no contexto de regras de negócio e simbolizam um contrato firmado entre duas ou mais classes, uma vez que são conceitualmente uma coleção de regras que devem ser seguidas pelas classes que a implementarem. As *interfaces* são capazes de definir um conjunto de métodos (apenas a assinatura do mesmo, seu nome e parâmetros) que devem ser implementados por determinadas classes, as quais criarão de fato os métodos assinados na *interface* [9].

No contexto da linguagem Java, por exemplo, cria-se *Interfaces* por meio da estrutura: *interface nome-da-interface { assinaturas }*. Para que uma classe realize uma *Interface*, é necessário adicionar a palavra reservada *implements* seguida do nome da *interface* a essa classe.

Esse conceito está presente no ambiente Robocode, mas não é facilmente perceptível. Quando cria-se uma classe especializada de *Robot*, como no fragmento de código abaixo, cria-se também o método *run* para o comportamento padrão do robô. Nesse caso, é válido ressaltar que a classe *Robot* especializa indiretamente a classe abstrata *_RobotBase* que, por sua vez, implementa a *Interface Runnable*, a qual define o método *run* que é visto na classe MeuRobo do exemplo.

```
1 // Classe nomeada MeuRobo
2 public class MeuRobo extends Robot
3 {
4
5     // Metodo definido pela Interface Runnable
6   public void run() {
7     // ... continuacao
8   }
```

```
9 }
```

Listagem 3.5 – Exemplo de Interfaces no Robocode.

3.2.9 Pacotes e Classes Utilitárias

Os pacotes chamados de utilitários são os que contêm e organizam as classes utilitárias dentro de um projeto em linguagens OO como Java. Por sua vez, uma classe utilitária pode ser definida por uma classe criada com o objetivo de fornecer recursos e funcionalidades às outras classes do projeto, assim, evitando-se a repetição de códigos por meio do compartilhamento. Normalmente, as classes utilitárias encontram-se dentro de um pacote chamado *util*. Comumente, as classes utilitárias são estáticas, não precisando ser instanciadas [9].

No ambiente Robocode, existem aplicações diretas do conceito de classes utilitárias. O fragmento de código abaixo apresenta uma chamada ao método *normalRelativeAngleDegrees*, o qual pertence à classe utilitária *Utils* do pacote *util* do próprio Robocode.

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends Robot
3  {
4      String trackName;
5      double gunTurnAmt;
6
7       // ... continuacao
8
9      // Metodo para quando atingir um inimigo
10     public void onHitRobot(HitRobotEvent e) {
11         if (trackName != null && !trackName.equals(e.getName()))
               {
12       out.println("Tracking " + e.getName() + " due to collision"
             );
13     }
14     trackName = e.getName();
15     gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (
          getHeading() - getRadarHeading()));
16     turnGunRight(gunTurnAmt);
17     fire(3);
18     back(50);
19   }
20 }
```

Listagem 3.6 – Exemplo de Classes Utilitarias no Robocode.

3.2.10 Streams

Segundo a documentação do Java 8 sobre *Stream*[1], os artigos da Oracle[2] e o livro de Matos [9], as *streams* são caracterizadas como uma cadeia de elementos, em outras palavras, podem ser entendidas como uma sequência de itens como, por exemplo, *bytes*, caracteres, ou algum outro tipo definido. Esse fluxo de dados é passível de aplicação de funções de agregação sequenciais ou paralelas de forma simples, o que facilita a programação e otimiza a execução do código gerado. Ademais, as *streams* também devem possuir um *source*, o qual provê os dados que são consumidos pela *stream* sobre demanda.

No ambiente Robocode pode ser encontrado o uso de *streams* em situações mais específicas. Um exemplo de aplicação dessas estruturas é o momento de realizar a leitura e gravação de dados em arquivos. No fragmento de código abaixo, deseja-se que o número de batalhas e *rounds* em que o robô participou seja atualizado em um arquivo. Para isso, utiliza-se o conceito de *streams*, mais especificamente os tipos *BufferedReader* (*stream* de caracteres com técnicas de armazenamento para antes de a operação ser aplicada), *FileReader* (*stream* específica para arquivos), *PrintStream* (*stream* de fluxo de saída de dados com tratamentos especiais para evitar exceções de entrada/saída) e *RobocodeFileOutputStream* (*stream* própria que equivale a *stream* de tratamento específico de arquivos de saída).

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends AdvancedRobot
3  {
4
5      // comportamento padrao do robo
6    public void run() {
7        int roundCount, battleCount;
8      try {
9        BufferedReader reader = null;
10       try {
11         reader = new BufferedReader(new FileReader(getDataFile("
               count.dat")));
12
13         roundCount = Integer.parseInt(reader.readLine());
14         battleCount = Integer.parseInt(reader.readLine());
15
16       } finally {
17         if (reader != null) {
18           reader.close();
19         }
```

[1] Disponível em: https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html
[2] Endereço: http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html

```
20        }
21      } catch (IOException e) {
22        roundCount = 0;
23        battleCount = 0;
24      } catch (NumberFormatException e) {
25        roundCount = 0;
26        battleCount = 0;
27      }
28      roundCount++;
29      if (!incrementedBattles) {
30        battleCount++;
31        incrementedBattles = true;
32      }
33      PrintStream w = null;
34      try {
35        w = new PrintStream(new RobocodeFileOutputStream(
              getDataFile("count.dat")));
36        w.println(roundCount);
37        w.println(battleCount);
38      } catch (IOException e) {
39        out.println("IOException trying to write: ");
40        e.printStackTrace(out);
41      } finally {
42        if (w != null) {
43          w.close();
44        }
45      }
46    }
47    // ... continuacao
48  }
```

Listagem 3.7 – Exemplo de Streams no Robocode.

3.2.11 Serialização e Persistência de Dados em Arquivos

O armazenamento de dados em arquivos para posterior utilização é um tópico importante em qualquer linguagem. No contexto da linguagem Java, esse armazenamento, quando se refere às classes e objetos, está intimamente ligado ao conceito de serialização e *streams*. A serialização é realizada pela realização da *interface Serializable*[3], a qual é responsável por indicar a possibilidade de conversão dos objetos dessa classe em uma sequência de *bytes* (*stream*) que podem ser transmitidos e armazenados. Para tanto, os objetos e suas

[3] Disponível em: https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html

características e estados são convertidos em uma *stream* que contém a assinatura da classe para que o objeto possa ser reconstruído corretamente em outro momento [9].

Durante a criação de robôs no ambiente Robocode não identificou-se a realização direta da *interface Serializable*, mas o conceito de persistência de dados em arquivos (sem a conversão de objetos em si) pode ser observada no fragmento de código a seguir, o qual indica a criação de uma *stream* de saída para um arquivo e a adição de dados sobre o número de batalhas e *rounds* nesse arquivo.

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends AdvancedRobot
3  {
4
5      // comportamento padrao do robo
6    public void run() {
7        int roundCount, battleCount;
8        // ... continuacao
9      PrintStream w = null;
10     try {
11       w = new PrintStream(new RobocodeFileOutputStream(
             getDataFile("count.dat")));
12       w.println(roundCount);
13       w.println(battleCount);
14     } catch (IOException e) {
15       out.println("IOException trying to write: ");
16       e.printStackTrace(out);
17     } finally {
18       if (w != null) {
19         w.close();
20       }
21     }
22   }
23   // ... continuacao
24 }
```

Listagem 3.8 – Exemplo de armazenamento de dados em arquivos no Robocode.

3.2.12 Atributos e Métodos

Atributos e métodos são conceitos fundamentais para a POO, visto que são intrínsecos às abstrações de Classes e Objetos. Os atributos nada mais são do que as características e marcas de estado dos objetos de uma determinada, ou da própria classe. Os métodos definem o comportamento dos objetos, em outras palavras, os métodos representam o que

é possível fazer com esses itens, podendo-se chamá-los para execução [9].

Naturalmente, no ambiente Robocode, utiliza-se ambos os conceitos durante a codificação de todos os robôs. O fragmento de código a seguir apresenta a criação direta de um atributo chamado energia que será único para cada objeto gerado. Além disso, modifica-se indiretamente os atributos de cores do robô por meio do método *setColors*. Por fim, o comportamento padrão do robô encontra-se no método chamado *run*.

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends Robot
3  {
4      double energia;
5      // comportamento padrao do robo
6    public void run() {
7      setColors(orange, blue, white, yellow, black);
8      while(true) {
9        ahead(100);
10       turnGunRight(360);
11       back(100);
12       turnGunRight(360);
13       energia = getEnergy();
14     }
15   }
16   // ... continuacao
17 }
```

Listagem 3.9 – Exemplo de Atributos e Metodos no Robocode.

3.2.13 Sobrecarga de Métodos

A abstração de sobrecarga de métodos é algo comum em programação com a linguagem Java. O conceito de sobrecarga consiste na criação de dois ou mais métodos que possuem o mesmo nome, mas que recebem parâmetros diferentes. Isso é um acontecimento comum quando há parâmetros que são considerados opcionais para um determinado método. Nesse caso, cria-se um método que receba os parâmetros opcionais e outro método, de mesmo nome, que não recebe esses parâmetros [9].

Um exemplo simples da aplicação dessa abstração no ambiente Robocode é o método *setColors*, o qual altera as cores do robô conforme os parâmetros fornecidos. Esse método é um método sobrecarregado da classe *Robot*. Os dois fragmentos de código a seguir seriam aceitos para chamar o método *setColores*, pois ele pode receber os parâmetros principais (cor do corpo, cor da arma e cor do radar) ou incluir os opcionais (cor do projétil e cor *scanner*) ao final.

```
1  // Classe nomeada MeuRobo - exemplo 1
2  public class MeuRobo extends Robot
3  {
4      // comportamento padrao do robo
5    public void run() {
6        setColors(orange, blue, white);
7      // ... continuacao
8    }
9    // ... continuacao
10 }
11
12 // Classe nomeada MeuRobo - exemplo 2
13 public class MeuRobo extends Robot
14 {
15     // comportamento padrao do robo
16   public void run() {
17       setColors(orange, blue, white, yellow, black);
18     // ... continuacao
19   }
20   // ... continuacao
21 }
```

Listagem 3.10 – Exemplo de Sobrecarga de Metodos no Robocode.

3.2.14 Sobrescrita

O conceito de *sobrescrita* (*overriding*) ocorre quando há o relacionamento de herança entre classes, pois é necessário que se tenha as abstrações de superclasse e subclasse para que o *overriding* ocorra. A sobreposição acontece quando há dois métodos com a mesma assinatura presentes ao mesmo tempo na superclasse e na subclasse. Quando isso ocorre, normalmente, o método correspondente da subclasse é o que será executado quando chamado [9].

No ambiente Robocode, o conceito de *overriding* é empregado com considerável frequência, mas pode não ser facilmente perceptível durante a codificação dos robôs. O fragmento de código a seguir indica a criação de um robô como subclasse da classe *Robot*. Essa superclasse realiza a *interface IBasicEvents* e, portanto, possui o método *onScannedRobot*. Contudo, a classe MeuRobo fez a sua própria versão especializada desse método (com a mesma assinatura). Assim sendo, o método *onScannedRobot* foi sobreescrito (*overriding*) pela subclasse.

```
1  // Classe nomeada MeuRobo
2  public class MeuRobo extends Robot
```

```
3 {
4     // comportamento ao ver outro robo
5   public void onScannedRobot(ScannedRobotEvent e) {
6     // ... continuacao
7   }
8   // ... continuacao
9 }
```

Listagem 3.11 – Exemplo de Overriding no Robocode.

3.2.15 Resumo da Análise

A análise apresentada nas subseções anteriores está sumarizada na Tabela 3. Cada conteúdo foi classificado em "Fortemente observável", "Fracamente observável" e "Não observável" conforme aparecem em maior ou menor grau durante a utilização de Robocode.

Tabela 3 – Elementos de POO observáveis no Robocode.

Conteúdo	**Fortemente observável**	**Fracamente observável**	**Não observável**
Classes e objetos	X		
Agregação			X
Herança	X		
Generalização	X		
Polimorfismo		X	
Encapsulamento	X		
Modificadores	X		
Exceções		X	
Programação genérica			X
Interfaces		X	
Composição			X
Pacotes e classes utilitárias	X		
Streams		X	
Serialização		X	
Persistência em arquivos		X	
Atributos	X		
Métodos	X		
Sobrecarga	X		
Sobrescrita		X	

Da análise dos dados apresentados na Tabela 3, percebe–se que alguns conceitos são bem perceptíveis, enquanto outros não são tão claramente observáveis. Temas como "conceitos fundamentais" e "conversão de tipos" não foram analisados por ser o primeiro genérico demais e o segundo muito específico para cada linguagem.

3.3 Análise por Alunos

Na seção anterior as funcionalidades de Robocode foram analisadas pelo autor do trabalho. Na presente seção é mostrada uma pesquisa de opinião realizada com discentes dos cursos de Sistemas de Informação e Ciência da Computação.

A fim de verificar se os alunos reconhecem os conceitos de orientação a objetos quando utilizam Robocode, foi aplicada uma atividade com sete estudantes. Propositalmente, foram selecionados alunos do primeiro ao quarto ano dos cursos, com o intuito de analisar se o conhecimento sobre programação influencia no reconhecimento das características que se quer analisar. Embora de períodos distintos, todos os alunos já tem conhecimentos básicos sobre classes e objetos.

Inicialmente, os alunos criaram o próprio robô, porém, sem saber previamente o que teriam que observar. Após criarem o robô e o utilizarem em uma batalha, os alunos responderam algumas perguntas, cujos temas foram os mesmos analisados pelo autor deste trabalho. As respostas disponíveis, em forma de alternativas, foram "sim", "não" e "não sei responder". Na Tabela 4 estão sumarizadas as respostas.

Tabela 4 – Resultado da análise pelos discentes.

Conceito	**Observado**	**Não Observado**	**Não sei responder**
Classes e Objetos	100 %	0 %	0 %
Atributos e Métodos	100 %	0 %	0 %
Agregação e Composição	42,9 %	0 %	57,1 %
Herança e Generalização	71,4 %	0 %	28,6 %
Polimorfismo	28,6 %	0 %	71,4 %
Encapsulamento e Modificadores	42,9 %	0 %	57,1 %
Tratamento de Exceções	42,9 %	14,3 %	42,9 %
Programação Genérica	57,1 %	14,3 %	28,6 %
Interfaces	85,7 %	14,3 %	0 %
Pacotes e Classes Utilitárias	71,4 %	0 %	28,6 %
Streams	28,6 %	0 %	71,4 %
Serialização	42,9 %	0 %	57,1 %
Sobrecarga	57,1 %	0 %	42,9 %
Sobrescrita	42,9 %	0 %	57,1 %

Além das perguntas de múltiplas escolhas, foi deixado espaço para que escrevessem livremente se consideravam a ferramenta útil durante o aprendizado de programação. Todas as opiniões foram favoráveis, destacando a simplicidade de Robocode. Um dos participantes disse que a aula poderia se tornar mais atrativa, enquanto outro observou que os conceitos de orientação a objetos podem ser observados mais realisticamente. Também é importante colocar a opinião de um dos participantes que destacou que o aprendizado pode ser mais divertido.

Os dados apresentados na Tabela 4 mostram duas situações antagônicas. Enquanto que 100% dos participantes concordaram que os conceitos de classes, objetos, atributos e métodos podem ser observados no Robocode, boa parte deles não souberam responder sobre encapsulamento e modificadores, por exemplo. Isto se deve ao fato de ainda não terem visto tais conceitos nas disciplinas.

3.4 Resultados e Discussão

Nas seções anteriores foram analisados alguns conceitos de orientação a objetos que podem ser trabalhados com Robocode. Foram realizadas duas análises, uma pelo autor do trabalho e outra com alunos de cursos de computação.

Em relação à análise pelo autor, percebeu–se que alguns conceitos são fortemente aplicáveis, enquanto outros, nem tanto. Há que se destacar que os conteúdos abordados em orientação a objetos variam entre os mais comuns (classes, atributos) aos menos comuns (serialização).

Já na análise realizada pelos alunos, há divergência entre os dados destes com os dados do autor. Isto se deve ao fato de nem todos os conceitos já terem sido vistos por eles. Mesmo com a divergência, boa parte dos dados das duas análises coincidiram.

Desta forma, pode–se concluir que a ferramenta pode ser útil como auxiliar para o ensino de programação orientada a objetos, porém de forma limitada. Esta limitação decorre do fato de que, apenas com o Robocode, o estudante poderia não ser capaz de entender os conceitos, uma vez que para construir um robô, são necessários alguns conhecimentos prévios de programação.

Robocode pode ser utilizado paralelamente ao estudo teórico dos conceitos de orientação a objetos. Um determinado conceito poderia ser estudado e, depois, aplicado com Robocode. Assim o aluno veria de forma menos abstrata a aplicação prática do tema.

4 CONSIDERAÇÕES FINAIS

O aprendizado de programação de computadores é considerado complexo por muitos estudantes, o que pode levar à reprovação nas disciplinas específicas bem como à evasão nos cursos de computação. Isto é preocupante, uma vez que a programação ocupa um papel importante em qualquer curso da área de computação.

Para diminuir as dificuldades, existem diversas ferramentas para auxílio durante a aprendizagem, tais como Alice, App Inventor, Scratch e Robocode. Este trabalho focou–se na análise desta última, no sentido de verificar se a ferramenta cobre os conteúdos tradicionalmente vistos nas disciplinas de programação orientada a objetos. Foram levantados os conteúdos ministrados nas disciplinas e implementado um exemplo para cada conteúdo. Com esta implementação, verificou–se se o conteúdo era visível durante a programação.

Neste contexto, verificou–se que Robocode pode ser utilizado como complemento durante o aprendizado de programação orientada a objetos. O programa consiste no desenvolvimento de robôs que devem batalhar entre si. Para a construção destes robôs é necessário empregar os conceitos de classe, objeto, herança, sobrecarga, dentre outros. A ideia é visualizar, na prática, os conceitos abstratos do paradigma OO. Com isto, minimiza–se um dos desafios em programação, que é a dificuldade em se lidar com a abstração.

Como trabalho futuro, pode–se investigar mais a fundo as funcionalidades de Robocode, inclusive no aspecto da inteligência artificial. Também é interessante aplicar em uma disciplina, para que os alunos utilizem a ferramenta durante o aprendizado, pois neste trabalho somente aspectos teóricos foram analisados.

REFERÊNCIAS

[1]

[2] L. R. Amaral, G. B. Silva, and E. Pantaleão. Plataforma robocode como ferramenta lúdica de ensino de programação de computadores. In *Anais do XXVI Simpósio Brasileiro de Informática na Educação (SBIE 2015)*, 2015.

[3] P. E. Battistella, G. Petri, C. G. Wangenheim, A. Wangenheim, and J. E. Martina. SORTIA 2.0: Um jogo de ordenação para o ensino de estrutura de dados. In *XII Brazilian Symposium on Information Systems*, 2016.

[4] J. Choi and M. Hannafin. Situated cognition and learning environments: Roles, structures and implications for design. *Educational Technology Research and Development*, 43(2):53–69, 1995.

[5] V. S. Fernandes and V. F. Junior. Linguagem de programação: Evasão e reprovação no instituto federal catarinense. In *V Simpósio de Integração Científica e Tecnológica do Sul Catarinense – SICT-Sul*, Araranguá - SC, 2016.

[6] V. P. Ferreira and J. R. Merlin. Ferramentas para ensino de programação. In *III Encontro de Integração UENP*, 2016.

[7] A. Gomes, C. Areias, J. Henriques, and A. J. Mendes. Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. *Revista Portuguesa de Pedagogia*, 42(2), 2008.

[8] C. S. Horstmann and G. Cornell. *Core Java: Volume 1 - Fundamentals*, volume 1. Prentice Hall, 9 edition, jul 2013.

[9] E. C. T. Mattos. *Programação de Softwares em Java*, volume 1. Digerati Books, São Paulo, 2007.

[10] C. C. Oliveira, J. W. Costa, and M. Moreira. *Ambientes informatizados de aprendizagem*, pages 53–69. Vozes, 2004.

[11] W. T. Oliveira. Um estudo sobre os problemas e dificuldades relacionados ao ensino e aprendizagem de programação., 2018. Monografia (Bacharel em Sistemas de Informação), UENP, Bandeirantes - PR.

[12] S. S. Prietch and T.A. Pazeto. Estudo sobre a evasão em um curso de licenciatura em informática e considerações para melhorias. In *Anais do WEIBASE*, Maceio, 2010.

[13] A. L. A. Raabe and J. M. C. Silva. Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos. In *XIII Workshop de Educação em Computação – WEI*, 2013.

[14] M. M. Rosa and L. M. M. Giraffa. O ensino de programação de computadores e EAD: uma parceria possível. In *XVII Congresso Internacional de Educação a Distância*, 2011.

[15] SESU-MEC. Diretrizes curriculares para os cursos da área de computação e informática. *MEC*, 2012.

[16] J. F. T. Suzuki, M. C. B. Steinle, and O. Battini. *TCC Elaboração e Redação*. Redacional Livraria, Londrina, 2009.

[17] T. R. Viegas, F. Y. Okuyama, M. Paravisi, and S. C. Bertagnolli. Uso das tics no processo de ensino-aprendizagem de programação. *Nuevas Ideas en Informatica Educativa TISE*, pages 780–785, 2015.