Aprendendo Conceitos de Orientação a Objetos Usando as Ferramentas Scratch e Snap!

Samuel da Silva Feitosa
Instituto Federal de Santa Catarina
Caçador, SC, Brasil
samuel.feitosa[at]ifsc.edu.br

Rafaela Lunardi Comarella
Instituto Federal de Santa Catarina
Florianópolis, SC, Brasil
rafaela.lunardi[at]ifsc.edu.br

ABSTRACT

The object-oriented programming paradigm serves as the basis for the most used programming languages nowadays, where large scale projects take advantage of certain characteristics, such as code reuse, encapsulation, better organization on the system design, and consequently the improvement on the maintainability of software in general. These features are intrinsic in the object-oriented paradigm. However, preparing professionals to work as systems analysts or developers for projects based in this paradigm has been shown difficult in academia, since it is necessary to understand a set of abstract concepts before starting the codification phase. In this context, this paper seeks to contribute with the study of two programming tools – Scratch and Snap! – developed exclusively for teaching computer programming, exploring them in order to teach object-oriented concepts in a visual and interactive mode, aiming to apply these concepts later in a traditional programming language such as Java, C#, or C++.

KEYWORDS

Programação Orientada a Objetos, Programação Visual, Pensamento Computacional

1 INTRODUÇÃO

Atualmente, o paradigma de Programação Orientada a Objetos (POO) predomina na lista das linguagens de programação mais populares [11]. Projetos de grande escala, onde as aplicações alcançam um alto nível de complexidade, vêm adotando este modelo de programação, uma vez que é possível explorar os seus conceitos-chave para aumentar a produtividade e qualidade no desenvolvimento de software.

É possível notar um crescimento acentuado na adoção de linguagens de programação que exploram os conceitos de Orientação a Objetos (OO) em projetos de tamanhos distintos, trazendo consigo a necessidade do ensino deste paradigma de desenvolvimento de software nos currículos dos cursos de qualificação, técnicos e superiores na área de computação. Considerando que, em geral, os cursos de programação iniciam o ensino a partir de algoritmos sequenciais, torna-se complicado a introdução do paradigma OO por parte dos professores, visto que os alunos precisam aprender a codificar e organizar de forma diferente ao que já estudaram [6].

Sendo assim, uma importante área de pesquisa diz respeito ao ensino de programação, na busca pelo desenvolvimento de técnicas e ferramentas que auxiliem no processo de ensino aprendizagem. Especificamente com relação à POO, alguns trabalhos [5, 7, 14] discutem formas de estimular e facilitar o aprendizado deste paradigma, uma vez que há uma dificuldade inerente dos estudantes dos cursos da área no que concerne ao aprendizado de diferentes modelos de programação.

Já existem algumas iniciativas desenvolvidas para auxiliar no aprendizado de programação, onde podem-se citar o site Code.org, que representa um movimento global para trazer os conceitos de computação ao alcance de todos, e a plataforma Scratch, que é um ambiente de programação visual concebido para auxiliar no processo de ensino de programação de computadores, dentre outros. Estas ferramentas adotam uma abordagem interativa para a aplicação dos mais diferentes conceitos. Mais diretamente conectado ao aprendizado de linguagens OO, existem algumas ferramentas similares, como por exemplo o Snap! que apresenta uma versão estendida do Scratch, GreenFoot que é um ambiente visual desenvolvido para propósitos educacionais, e Alice que também é um ambiente de programação baseado em blocos, permitindo além do ensino de programação de modo geral, também a aplicação dos conceitos do paradigma orientado a objetos enquanto desenvolvendo aplicações.

Neste contexto, este trabalho pretende apresentar uma descrição dos conceitos de orientação a objetos que podem ser aplicados e desenvolvidos através da ferramenta Scratch, em contraste com a sua extensão Snap!, comparando as suas funcionalidades e apresentando formas de aplicá-las como forma de facilitar o processo de ensino aprendizagem de POO. A partir de pesquisas anteriores [14] é possível observar que a utilização destas ferramentas interativas auxiliam no processo de aprendizagem dos conceitos de programação de computadores por parte de pessoas que não possuem conhecimento nesta área, pretendendo-se replicar este método também para conceitos mais avançados como o de POO.

Especificamente, este artigo apresenta as seguintes contribuições:

- Um catálogo contendo a descrição dos principais conceitos de OO, apresentando as suas principais funcionalidades e discutindo possíveis aplicações.
- Uma comparação entre as funcionalidades apresentadas nas ferramentas Scratch e Snap!, considerando as possíveis aplicações em sala de aula para os conceitos de OO.

Os demais capítulos estão organizados da seguinte forma: no capítulo 2 são introduzidos os principais conceitos relativos a POO, apresentadas as ferramentas Scratch e Snap! e discutido brevemente alguns trabalhos relacionados. No capítulo 3, são discutidos os procedimentos metodológicos desta pesquisa. No capítulo 4 apresentamos o catálogo dos conceitos de OO relacionando os mesmos de forma comparativa com as ferramentas educacionais apresentadas anteriormente. E finalmente, no capítulo 5 são apresentadas as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

O ensino de programação de computadores tem motivado diversas discussões nas últimas décadas no mundo inteiro, principalmente por ser considerado como tema elementar nos currículos, com uma grande importância para diferentes cursos técnicos, além da área de computação. Diversos países estão adaptando os currículos das escolas de nível fundamental e médio para introduzir conceitos de pensamento computacional [2]. Como existem diferentes paradigmas de programação, o problema com o ensino de orientação a objetos também tem sido alvo de pesquisas na área.

Diversas ferramentas para o ensino de programação têm sido lançadas nos últimos anos, as quais geralmente compartilham dos mesmos objetivos: criar um contexto de programação que engaje os estudantes (geralmente envolvendo jogos), e diminuir as dificuldades iniciais para com a sintaxe das linguagens, permitindo foco nos conceitos de programação. Dentre as mais conhecidas, além das ferramentas alvo deste trabalho, podem-se citar as mais recentes variações da ferramenta Logo que incluem NetLogo [13], StarLogo [10], a plataforma Karel o Robô e suas variações [9], o portal Code.org[1], a ferramenta Alice [3] e o ambiente de desenvolvimento Greenfoot [7].

A maioria destas ferramentas é voltada para um público específico, visando sempre desenvolver as capacidades cognitivas para o pensamento computacional, mais especificamente com relação a orientação a objetos, algumas variações da plataforma Karel, como por exemplo JKarel [12] e Karel++ [1] foram desenvolvidas para o ensino das linguagens Java e C++ respectivamente, oferecendo uma sintaxe que facilita a transição entre o sistema e a linguagem de programação.

No caso das ferramentas Alice e Greenfoot, ambas têm por objetivo o ensino de programação para pessoas sem experiências anteriores, introduzindo conceitos de programação orientada a objetos baseada em classes, com ênfase na linguagem Java e seus conceitos. Greenfoot fornece aos seus usuários mecanismos avançados para desenvolver computação de alta performance, e estender o seu sistema base (o que também é possível com o Snap!). Já a ferramenta Alice oferece um ambiente de desenvolvimento que suporta gráficos 3D, o que pode ser visto como mais atuais pelos alunos que estão acostumados com jogos modernos.

Aprender programação não é uma tarefa trivial. Não basta conhecer as construções internas da linguagem de programação e ter ideia de como utilizá-las. É necessário compreender a maneira de se pensar nas soluções de acordo com o paradigma de programação que está sendo estudado. O pensamento computacional é fundamentalmente estruturado em conceitos de sequência, funções, relações ou equações, que geralmente são ensinados nos primeiros contatos com programação de computadores nos cursos tradicionais. Entretanto, para o caso da POO, não existe uma teoria bem estabelecida ou framework conceitual que demonstra como aprender seus conceitos.

Desta forma, nesta seção serão apresentadas as principais características das linguagens que implementam o paradigma orientado a objetos, bem como as ferramentas de desenvolvimento visual a serem exploradas para a aplicação e o ensino de conceitos de orientação a objetos.

[1]CODE.ORG: Disponível em https://code.org/ .

2.1 Orientação a Objetos

Uma das razões pelas quais POO tornou-se largamente aceita é que os conceitos de OO são parecidos com as nossas percepções naturais do mundo real. O principal fundamento da POO é fazer com que os programas reflitam ao máximo a realidade que será tratada. Deste modo, torna-se mais fácil entender o que está descrito nos programas. Isto se deve ao fato de que os seres humanos são, desde o princípio, acostumados e treinados na percepção do que está acontecendo no mundo real. Quanto mais próximo for para aplicar este pensamento em programação, mais fácil será para escrever e entender programas [8].

Segundo Madsen, Mø-Pedersen e Nygaard [8] o conceito de orientação a objetos pode ser entendido informalmente como sendo: "a execução de um programa diz respeito a um modelo físico, simulando o comportamento de uma parte real ou imaginária do mundo". De fato, quando se escreve código neste paradigma computacional, diversos objetos são modelados, e estes interagem entre si para resolver um determinado problema.

Em comparação com a programação procedural, a POO pode ser descrita como o privilégio dos dados sobre as ações. Na programação procedural o programador aborda o problema decompondo o sistema em uma série de ações, frequentemente chamadas de funções. Em POO o programador primeiramente decompõe o problema em sub-rotinas de código reusável, evitando a replicação de código de mesma funcionalidade em diferentes partes do sistema [5].

A seguir serão apresentados brevemente os principais conceitos de POO [4]:

Abstração. Significa decompor um sistema complicado em suas partes fundamentais e descrevê-las em uma linguagem simples e precisa.

Objeto. É uma combinação específica de dados e dos métodos capazes de processar e comunicar estes dados. Também são chamados de instâncias de classes.

Classe. É uma descrição de objetos com o mesmo conjunto de atributos, comportamentos, ou relacionamentos, ou seja, elas definem os tipos dos objetos.

Encapsulamento. Estabelece que os diferentes componentes de um sistema de software não devem revelar detalhes de suas respectivas implementações, fornecendo ao programador liberdade na implementação dos detalhes internos do sistema.

Atributos. São os dados de um objeto, também conhecidos como variáveis de instância.

Métodos. São as operações que podem atuar sobre os dados e que expressam as mensagens às quais os objetos respondem. Definem o comportamento dos objetos.

Herança. Permite projetar classes de forma genérica que podem ser especializadas em classes mais particulares, oferecendo uma estrutura hierárquica e modular para a reutilização de código.

Polimorfismo. Refere-se à habilidade de uma variável de objeto de assumir formas diferentes.

Exceções. São eventos inesperados que ocorrem durante a execução de um programa, como por exemplo, uma condição de erro ou uma simples entrada inesperada. Estas são lançadas em trechos de código que detectam a condição inesperada e capturadas por trechos de códigos capazes de tratá-las de alguma forma.

Muitas linguagens de programação populares hoje em dia são consideradas orientadas a objetos, pois elas incluem as funcionalidades listadas acima, permitindo ou até incentivando o programador a projetar o seu código em um estilo orientado a objetos. Alguns exemplos de linguagens neste paradigma são: Java, C# e C++, as quais são estaticamente tipadas, e Javascript, Python e Ruby, as quais são dinamicamente tipadas [5].

Na seção a seguir, são apresentadas as ferramentas de programação visual que motivaram o presente estudo.

2.2 Programação Visual

Programação visual baseada em blocos é uma área relativamente nova no que tange as linguagens de programação e ambientes desenvolvidos para o aprendizado de conceitos de computação. Nestes ambientes, os estudantes podem construir programas através de blocos pré-definidos utilizando apenas o mouse para encaixar instruções, recebendo um feedback visual indicando que uma determinada construção é válida ou não.

Neste artigo será apresentada especificamente a ferramenta Scratch e sua extensão chamada Snap! com o objetivo de aplicar conceitos de orientação a objetos nas mesmas.

2.2.1 ***A Ferramenta Scratch***. É um ambiente visual[2] desenvolvido para facilitar o aprendizado de conceitos de programação, tendo como alvo principal crianças e adolescentes. A partir desta ferramenta, usuários podem criar projetos e utilizar uma interface que fornece blocos com funcionalidades pré-definidas, permitindo a construção de programas, jogos ou animações a partir da junção destes blocos. A possibilidade de compartilhamento de projetos também se mostra interessante no sentido de discutir diferentes soluções por uma comunidade de usuários que pode fomentar o aprendizado em conjunto.

Esta ferramenta desenvolvida pelo grupo Lifelong Kindergarden no MIT Media Lab, possui atualmente traduções para mais de 70 idiomas diferentes e vem sendo utilizada por estudantes em casa, nas escolas e em grupos de estudo em diferentes partes do mundo. Geralmente tem seu uso incentivado por professores em disciplinas de programação, ciência da computação, e pensamento computacional, podendo também ser utilizada em diferentes áreas.

2.2.2 ***A Extensão Snap!***. É também uma linguagem de programação visual[3] baseada na combinação de blocos com funcionalidades pré-definidas. É uma reimplementação estendida do Scratch, que como funcionalidade adicional, permite a criação e customização de blocos de código, além de possibilitar a utilização de alguns conceitos de programação de linguagens modernas. A partir das funcionalidades adicionadas, a ferramenta se mostra adequada para a introdução de conceitos de ciência da computação seja para alunos do ensino básico, bem como os de nível superior.

[2]SCRATCH: Disponível em https://scratch.mit.edu/.
[3]SNAP!: Disponível em https://snap.berkeley.edu/.

3 PROCEDIMENTOS METODOLÓGICOS

O presente estudo constitui-se de uma pesquisa bibliográfica exploratória com relação aos conceitos do paradigma de orientação a objetos e a forma com que os mesmos podem ser abordados no processo de ensino aprendizagem para estudantes da área de computação. Também apresenta ferramentas visuais que estão sendo utilizadas no ensino de programação e que podem ser úteis para o ensino de orientação a objetos. A partir da análise do aparato conceitual e ferramental pesquisado, propõe-se a aplicação deste paradigma através de uma abordagem experimental nas ferramentas visuais estudadas, o que possibilita o estudo de orientação a objetos de forma diferente da usual, por meio de uma abordagem lúdica.

Esta catalogação de conceitos de programação para com uma ferramenta visual pode ser útil na obtenção de várias habilidades relacionadas com a resolução de problemas, permitindo que os conhecimentos adquiridos inicialmente em computação, possam também ser aplicados em situações cotidianas. Além disso, por meio da utilização destas ferramentas visuais é possível abordar problemas complexos de maneira mais simples que a usual, motivando e estimulando os alunos a um aprendizado autônomo e prático.

4 RESULTADOS E DISCUSSÃO

As novas ferramentas visuais de ensino de linguagens de programação têm se mostrado efetivas quando utilizadas na introdução de conceitos tanto de orientação a objetos quanto da funcionalidade de um programa de computador. Também, a impossibilidade de se cometer erros de sintaxe mostra-se de grande valia para iniciantes, fazendo com que os mesmos possam dar mais atenção aos componentes e ao fluxo de execução do código. Além disso, pela ferramenta trazer diferentes blocos de controle de forma visual (como laços de repetição e estruturas condicionais), é possível demonstrar aos alunos as associações necessárias que serão realizadas posteriormente em uma linguagem de programação baseada em texto.

Nas próximas seções serão apresentados diversos conceitos de orientação a objetos que podem ser ensinados através das linguagens de programação visual Scratch e Snap!.

4.1 Orientação a Objetos com Scratch

A criação ou instanciação de objetos pode ser facilmente demonstrada através da adição de um personagem na tela ou de um determinado elemento gráfico, como demonstrado na Figura 1.

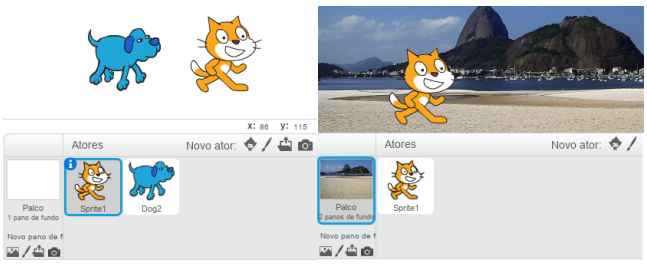

Figura 1: Instanciação de objetos no Scratch.

É importante frisar que um plano de fundo também é visto como um objeto e, portanto, pode ser utilizado para demonstrar conceitos abstratos, como por exemplo instanciação de objetos e acesso a atributos.

Outro conceito de OO que pode ser demonstrado de forma simples através do Scratch diz respeito aos atributos de um objeto. Cada personagem (ou elemento gráfico) tem propriedades como nome, coordenadas, direção, estilo de rotação, etc., conforme pode ser visto na Figura 2.

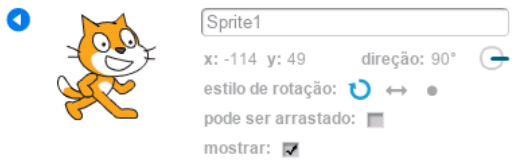

Figura 2: Atributos de um objeto no Scratch.

A apresentação dos atributos pode ser introduzir a discussão a respeito do encapsulamento de informações, sendo que cada objeto tem suas próprias informações e estas podem ser acessadas de diferentes formas.

A maioria das propriedades dos objetos podem ser modificadas através de algum bloco pré-definido, permitindo a introdução do conceito de métodos de forma simples e gradual. Na Figura 3 pode-se visualizar alguns blocos que possibilitam a modificação de propriedades de um objeto.

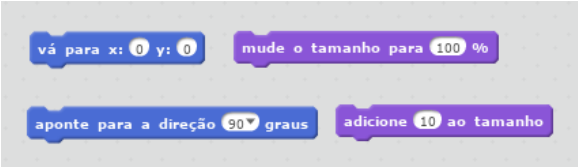

Figura 3: Operações simples no Scratch.

Por meio da utilização destes blocos simples, é possível reforçar o conceito de encapsulamento de informações de um objeto, onde um atributo é acessado ou modificado através de um método. Também é possível o desenvolvimento de métodos mais complexos, através da construção de scripts com operações condicionais e laços de repetição para realizar alguma atividade. A Figura 4 apresenta alguns exemplos utilizando estas operações.

Figura 4: Operações condicionais e de repetição no Scratch.

É importante demonstrar aos alunos que cada personagem possui a sua própria linha de execução, ou seja, cada objeto possui seus próprios atributos e métodos, os quais podem (ou não) ser passíveis de execução de acordo com a programação realizada.

Quando se desenvolve um software utilizando o paradigma de orientação a objetos, diversos componentes são desenvolvidos separadamente, e a inter-relação entre eles é o que define o sistema como um todo. Sendo assim, os objetos devem ter a capacidade de comunicar-se entre si através da troca de mensagens. Em uma linguagem de programação OO, a troca de mensagens se dá basicamente através da invocação de métodos de um objeto que compõe ou agrega outro objeto.

A ferramenta Scratch também possui um mecanismo elaborado de troca de mensagens entre os objetos, que inclui o envio de mensagens de forma sincronizada ou não, e o recebimento de mensagens por determinado objeto. A Figura 5 apresenta os principais blocos responsáveis por estas atividades.

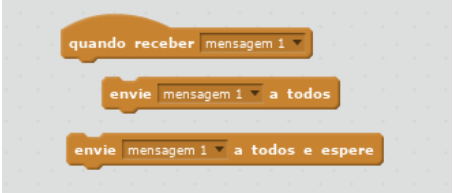

Figura 5: Troca de mensagens no Scratch.

É importante destacar neste ponto que os objetos no Scratch não são relacionados diretamente como em uma linguagem tradicional (por meio de composição ou agregação). Nesta ferramenta, um determinado objeto pode enviar uma mensagem a todos os outros objetos disponíveis no sistema, e o objeto responsável por tratar essa mensagem deve ter um método (bloco de código) associado, responsável por tratar a mensagem recebida. Sendo assim, a invocação de um determinado método de um objeto não é feita diretamente, e sim por meio deste sistema de troca de mensagens. Seguindo a funcionalidade de uma linguagem de programação OO, a forma de simular mais precisamente uma chamada de método é através do bloco que envia uma mensagem e aguarda o seu processamento. Por se tratar de uma diferença substancial entre a ferramenta Scratch e uma linguagem tradicional, o professor deve considerar explanações extra-ferramenta para definir tais conceitos.

O Scratch possui também outras funcionalidades além das apresentadas que podem ser úteis para ensinar diferentes conceitos de programação (não necessariamente de orientação a objetos). Por estar fora do escopo principal deste trabalho, serão apenas listadas algumas destas funcionalidades.

Ao utilizar essa ferramenta é possível introduzir a discussão de conceitos como “eventos” e “processamento paralelo”, que também podem fazer parte de linguagens de programação em outros paradigmas. Os blocos associados a eventos no Scratch são executados quando determinado evento ocorre no sistema. Alguns destes eventos podem ser vistos na Figura 6.

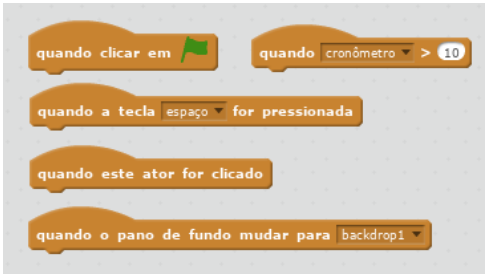

Figura 6: Eventos no Scratch.

Com a apresentação destes eventos é possível fazer um paralelo com o desenvolvimento de interfaces gráficas, intercomunicação de sistemas, alguns conceitos de bancos de dados, etc.

Já com relação ao processamento paralelo, é importante demonstrar aos alunos o conceito de threads, uma vez que no Scratch cada script é considerado um processo individual. Nesta mesma questão, podem ser explorados os conceitos de trocas de mensagens síncronas e assíncronas. A Figura 7 demonstra dois scripts que executam em paralelo no ambiente interativo.

Figura 7: Script com processamento paralelo.

O Scratch é uma ferramenta que possibilita a explanação de diferentes conceitos de OO, sendo que alguns podem ser demonstrados de forma muito simples, e outros com alguma certa dificuldade. Entretanto, existem alguns conceitos de OO que não são cobertos pelo Scratch (ou atendidos parcialmente ou de forma complexa), e sendo assim, buscou-se uma alternativa para tal através da ferramenta Snap!, conforme apresentado na seção seguinte.

4.2 Orientação a Objetos com Snap!

A maioria das linguagens de programação orientadas a objetos utiliza classes e instância de classes quando trata da criação de objetos. Uma classe define um tipo particular de objeto, e uma instância é o objeto propriamente dito de um tipo. No Snap! a instanciação de objetos é abordada de forma diferente, utilizando a abordagem conhecida por prototipação, onde não há distinção entre classes e instâncias. Sendo assim, cada personagem adicionado pode ser visto como a instância de determinado objeto de uma classe, e este pode ser replicado para novas cópias (clones ou filhos) que compartilham do mesmo comportamento, porém podendo ter suas próprias informações. A criação de um novo objeto a partir de um objeto existente, pode ser realizada de duas formas: clicar com o botão direito no personagem, e selecionar a opção para "clonar" o objeto; ou utilizar o bloco pré-definido para criação de clones no menu "Controle", conforme visto na Figura 8.

Figura 8: Criação de objetos através de prototipação no Snap!.

É possível simular facilmente a hierarquia de classe/instância ocultando o personagem que serviu de protótipo. Este formato de prototipação faz parte do princípio de ambas as ferramentas (Scratch e Snap!), onde todos os itens de um projeto devem ser visíveis no cenário de programação, diferenciando da forma da maioria das linguagens OO, onde primeiramente é criado um modelo abstrato e invisível (classe) que deve ser programada antes que qualquer objeto concreto possa ser criado.

De forma similar ao Scratch, o Snap! possui os personagens, que são vistos como objetos naturais. Cada personagem pode ter, além dos seus atributos padrão, variáveis. Entretanto, os objetos (planos de fundo, personagens, aparência, sons, etc.) no Scratch são menos versáteis do que no Snap!, onde todo e qualquer objeto na linguagem é visto como um dado de primeira classe, o qual pode ser atribuído a uma variável, pode ser um elemento de uma lista, e assim por diante. A Figura 9 mostra um exemplo onde uma variável foi criada para armazenar a lista de amigos de um determinado personagem (neste caso todos os personagens vizinhos), e dois comandos para apresentar cada um dos elementos da lista.

Figura 9: Dados de primeira classe no Snap!.

Como pode-se notar, objetos podem ser atribuídos e apresentados na tela como quaisquer outros tipos de dados. Isto é importante, pois simula de forma mais próxima ao que ocorre em uma linguagem de programação de propósito geral.

O Snap! possui um mecanismo aprimorado para troca de mensagens em comparação ao Scratch. Além de possibilitar as mesmas formas de comunicação do Scratch (apresentadas na seção anterior), ele também permite a troca de mensagens a objetos individuais. Por exemplo, é possível, a partir de um determinado objeto, executar um código que instrua outro objeto a executar determinada tarefa. Na Figura 10 é demonstrado duas formas para solicitar que determinado bloco execute o comando de movimento.

Figura 10: Envio de mensagens para um objeto individual no Snap!.

O bloco "execute" recebe dois argumentos, sendo o primeiro a ação a ser executada, e o segundo a qual objeto esta mensagem deve ser enviada. Na parte esquerda da figura acima, pode-se notar que está sendo solicitada a execução do movimento para o personagem de nome "Ator". Na parte direita da imagem, o comando de execução é combinado com um laço de repetição, aplicando-o a todos os personagens vizinhos. Existem também outros blocos de ação que podem ser combinados para realizar efeito similar.

Também, o Scratch fornece uma maneira de obter informações sobre certas propriedades de outros objetos, através de um bloco para tal na paleta "Sensores". O Snap! estende este mecanismo para qualquer propriedade de um personagem, o que apresenta funcionalidade similar a linguagens de programação tradicionais.

Da mesma forma que a função "clone" pode ser utilizada para demonstrar a instanciação de objetos a partir de um protótipo, o mesmo mecanismo é utilizado para simular herança. Quando um clone é criado, ele herda as propriedades do objeto ancestral. Nestas propriedades estão incluídos os scripts, blocos customizados, variáveis, atributos, etc. Cada uma destas propriedades pode ou não ser compartilhada entre pai e filhos. Deste modo, quando determinado objeto filho recebe a solicitação para realizar uma operação, é verificado se existe um bloco correspondente a ação solicitada, e caso contrário, a requisição é repassada ao objeto pai. Em linguagens de programação OO, este mecanismo é apresentado como sobrescrita de métodos em uma hierarquia de herança.

O Snap! também oferece a possibilidade de simular polimorfismo em suas definições, uma vez que a partir desta ferramenta é possível definir seus próprios blocos customizados. Sendo assim, é possível em um personagem pai fazer uma chamada a um determinado bloco customizado, e personagens filhos definirem estes blocos de forma individual, criando métodos polimórficos, uma vez que, diferentes trechos de código são executados a partir dos filhos, mesmo que estes compartilhem o mesmo script inicial.

O Snap!, assim como o Scratch, possui inúmeras outras funcionalidades que podem ser exploradas tanto no ensino de programação OO, quanto no ensino de pensamento computacional de forma geral. Um aspecto interessante com relação ao Snap! é que este permite ao programador a criação dos próprios blocos de programação, deste modo permitindo a expansão da linguagem da forma que for mais conveniente para os projetos.

Considerando este aspecto, na própria plataforma visual é possível "importar" bibliotecas já desenvolvidas para auxiliar no desenvolvimento, trazendo consigo um dos principais benefícios da programação OO que é a reusabilidade de código. Nestas bibliotecas é possível encontrar blocos mais elaborados para lidar com aspectos importantes de programação, como por exemplo: tratamento de exceção, laços de repetição diferentes do padrão, manipulação de listas, etc.

A Figura 11 apresenta à esquerda como importar bibliotecas disponíveis no sistema Snap!. Na parte direita são apresentados alguns dos operadores disponíveis após a importação.

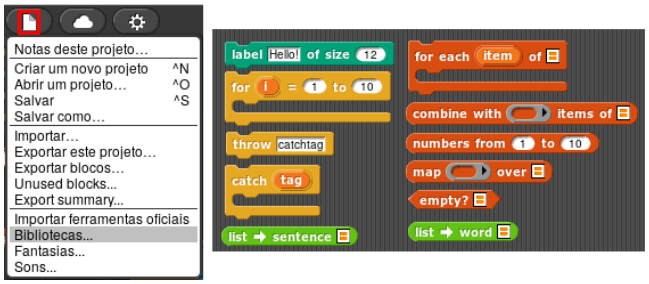

Figura 11: Operadores importados das bibliotecas Snap!.

Considerando que no Snap! todos os componentes da linguagem podem ser vistos como dados, é possível também trabalhar com funções anônimas (expressões lambda) e funções de alta ordem. Na figura acima podemos ver o bloco "map" que permite executar uma função em todos os elementos de uma lista. Esta funcionalidade também é importante quando considerando projetos OO, uma vez que a grande maioria das linguagens de programação modernas fornecem mecanismos com funcionalidade similar.

4.3 Comparativo entre Scratch e Snap!

Após apresentar como simular diversos conceitos de programação nas ferramentas Scratch e Snap!, sumarizou-se na Figura 12 as principais funcionalidades das linguagens OO e a sua relação de adesão para com as ferramentas estudadas.

Figura 12: Comparativo de adesão entre Scratch e Snap! para com conceitos de OO.

Este quadro demonstra que ambas as ferramentas nos possibilitam o ensino de diversos conceitos de POO, entretanto a ferramenta Scratch apresenta limitações para com o uso de atributos, uma vez que podemos utilizar apenas atributos pré-definidos, enquanto que o Snap! permite estender os atributos originais dos objetos utilizados em um projeto. Ainda de acordo com o quadro, apenas o Snap! possui funcionalidades que permitem simular herança e polimorfismo, além de ser o único que permite a importação de bibliotecas. Por fim, o quadro sumariza em sua última linha as diversas outras funcionalidades mencionadas no texto, ambos atendidos parcialmente, indicando que as linguagens visuais estudadas podem auxiliar no processo de ensino aprendizagem de OO, porém com as devidas ressalvas e limitações.

5 CONSIDERAÇÕES FINAIS

Este artigo apresentou um estudo com duas ferramentas de programação visual desenvolvidas exclusivamente para o ensino de programação de computadores, vinculando suas funcionalidades de forma a apresentar os conceitos de programação orientada a objetos de forma interativa. Este estudo foi motivado devido a dificuldade no ensino destes conceitos em diferentes níveis de atuação (nível médio ou superior), e pela falta de metodologias para facilitar este processo. Obviamente, este trabalho não cobre todos os aspectos de uma disciplina de orientação a objetos e serve apenas um ponto de entrada para a exploração destas ferramentas em sala de aula, podendo ser aplicada gradativamente em disciplinas específicas em cursos nesta área.

De qualquer modo, baseado neste estudo é possível vislumbrar a utilidade destas ferramentas como auxiliares no processo de ensino-aprendizagem, uma vez que permitem que os alunos visualizem os conceitos de forma concreta, evitem erros sintáticos, e executem visualmente os passos de uma determinada aplicação, facilitando a compreensão dos conceitos abstratos de programação OO. Além disso, pode-se perceber que a ferramenta Scratch tem um suporte limitado para o ensino deste conceitos, uma vez que foi desenvolvida com um propósito de ensino mais geral, e a ferramenta Snap!, apesar de ser mais complexa inicialmente, permite ao programador explorar diversos conceitos que podem ser aplicados posteriormente em uma linguagem de programação orientada a objetos.

Como trabalhos futuros, as técnicas apresentadas neste texto podem ser expandidas para cobrir um maior número de conceitos de orientação a objetos, pode ser desenvolvida uma metodologia de aplicação em sala de aula das ferramentas apresentadas, além da experimentação das mesmas para verificar o potencial de melhoria de compreensão por parte dos alunos.

6 AGRADECIMENTOS

Agradecimentos especiais aos professores Eli Lopes da Silva e Igor Thiago Marques Mendonça pelas valiosas contribuições para a melhoria da apresentação dos resultados deste artigo.

REFERÊNCIAS

[1] Joseph Bergin, Jim Roberts, Richard Pattis, and Mark Stehlik. 1996. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[2] Chen Xiajian, Wang Danli, and Wang Hongan. 2011. Design and implementation of a graphical programming tool for children. In *2011 IEEE International Conference on Computer Science and Automation Engineering*, Vol. 4. 572–576. https://doi.org/10.1109/CSAE.2011.5952915

[3] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated Transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*. ACM, New York, NY, USA, 141–146. https://doi.org/10.1145/2157136.2157180

[4] Paul J Deitel and Harvey M Deitel. 2016. *Java: como programar* (10th ed.). Pearson.

[5] Hampus Gunnrup and Pooriya Balavi. 2017. Children Learning Object Oriented Programming: A Design Science Study. (2017). Monografia (Graduação em Engenharia e Gerenciamento de Software), Universidade de Gotemburgo, Gotemburgo, Suécia.

[6] Michael Kölling. 1999. The Problem of Teaching Object-Oriented Programming, Part I: Languages. *JOOP* 11 (1999), 8–15.

[7] Michael Kölling. 2010. The Greenfoot Programming Environment. *Trans. Comput. Educ.* 10, 4, Article 14 (Nov. 2010), 21 pages. https://doi.org/10.1145/1868358.1868361

[8] Ole Lehrmann Madsen, Birger Mø-Pedersen, and Kristen Nygaard. 1993. *Object-oriented Programming in the BETA Programming Language*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[9] Richard E. Pattis. 1981. *Karel the Robot: A Gentle Introduction to the Art of Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[10] StarLogo. 2019. MIT Scheller Teacher Education Program. https://education.mit.edu/project/starlogo-tng/. (10 2019). Accessed: 2019-10-15.

[11] tiobe.com. 2019. TIOBE Index. https://www.tiobe.com/tiobe-index/. (09 2019). Accessed: 2019-09-11.

[12] H. Wellenius. 2019. JKarel Documentation and Software. http://www.cs.tufts.edu/comp/10F/JKarel.htm. (10 2019). Accessed: 2019-10-15.

[13] U. Wilensky. 2019. NetLogo. http://ccl.northwestern.edu/netlogo/. (10 2019). Accessed: 2019-10-15.

[14] Lu Yan. 2009. Teaching Object-Oriented Programming with Games. In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations (ITNG '09)*. IEEE Computer Society, Washington, DC, USA, 969–974. https://doi.org/10.1109/ITNG.2009.13