INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO TRIÂNGULO MINEIRO – *CAMPUS* UBERABA
Programa de Pós-Graduação em Educação Tecnológica
Mestrado Profissional em Educação Tecnológica

FRANCIELLI BARBARA PINTO

UMA PROPOSTA *DESPLUGADA* E MULTIPARADIGMÁTICA PARA O ENSINO DE PROGRAMAÇÃO NA EDUCAÇÃO BÁSICA

Uberaba
2019

FRANCIELLI BARBARA PINTO

UMA PROPOSTA *DESPLUGADA* E MULTIPARADIGMÁTICA PARA O ENSINO DE PROGRAMAÇÃO NA EDUCAÇÃO BÁSICA

Dissertação apresentada ao Programa de Pós-Graduação em Educação Tecnológica - curso de Mestrado Profissional em Educação Tecnológica do Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro – Campus Uberaba, como requisito parcial para obtenção do título de Mestre em Educação Tecnológica.

Linha de Pesquisa: Tecnologias da Informação e Comunicação (TICs), inovação tecnológica e mudanças educacionais

Orientador: Prof. Dr. André Souza Lemos

Uberaba
2019

Ficha Catalográfica elaborada pelo Setor de Referência do IFTM – Campus Uberaba-MG

P658u

Pinto, Francielli Bárbara
Uma proposta desplugada e multiparadigmática para o ensino de programação na educação básica / Francielli Bárbara Pinto – 2019.
76 f.: il.

Orientador: Prof. Dr. André Souza Lemos
Dissertação (Mestrado Profissional em Educação Tecnológica) Instituto Federal do Triângulo Mineiro- Campus Uberaba- MG, 2019.

1. Ensino de programação. 2. Educação básica. 3. Pensamento computacional. I. Lemos, André Souza. II. Título.

CDD 371.33

Francielli Barbara Pinto

Uma proposta desplugada e multiparadigmática para o ensino de programação na Educação Básica

FOLHA DE APROVAÇÃO DEFESA DISSERTAÇÃO

Data da aprovação: 13/09/2019

MEMBROS COMPONENTES DA BANCA EXAMINADORA:

Presidente e orientador: **Prof. Dr. André Souza Lemos**
Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro – IFTM
Campus Uberaba

Membro Titular **Prof. Dr. Adriano Eurípedes Medeiros Martins**
Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro – IFTM
Campus Uberaba

Membro Titular **Profa. Dra. Crícia Zilda Felício Paixão**
Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro – IFTM (Campus Uberlândia Centro)

Local: sala 104 do IFTM Campus Uberlândia Centro

AGRADECIMENTOS

Agradeço, primeiramente, a Deus por tudo o que tem me proporcionado, por me guiar, principalmente durante a tantas idas a Uberaba e voltas para casa, por cuidar de mim e das pessoas próximas e por sempre iluminar meu caminho.

Agradeço a minha mãe por tudo que fez e faz pelos filhos, por colocar a educação sempre em primeiro lugar, por me mostrar o caminho do bem e da luta. Agradeço ao meu pai por ser um pai tão amoroso e presente, e por ter me acompanhado tantas vezes a Uberaba, teve papel fundamental nessa jornada. Se hoje celebro conquistas, muito é graças a eles.

Ao meu orientador e professor, Dr. André Souza Lemos, por se fazer sempre à disposição para me orientar, tirar minhas dúvidas, para reuniões pelo Skype e conversas pelo Messenger e Whatsapp, sempre de maneira muito serena, motivadora e leve. Um exemplo de orientador a ser seguido. Pelas inúmeras contribuições em minha formação e em meu trabalho, por toda aprendizagem proporcionada, por compartilhar um pouco de sua sabedoria.

A Ana Helena pelo companheirismo inigualável, seja nos bons e maus momentos. Companheira para todas as horas, por me permitir compartilhar todas as angústias e alegrias.

A todos os amigos e familiares que fizeram parte de certa forma desta caminhada, tornando-a mais leve e com mais sentido. Em especial ao Carlos Henrique (Caca) por se fazer sempre presente e pelos inúmeros incentivos e a minha prima e companheira de trabalho, Bruna, por estar sempre disposta a auxiliar nas mais diversas questões.

Aos professores do Programa de Pós-Graduação em Educação Tecnológica - curso de Mestrado Profissional em Educação Tecnológica do IFTM – Campus Uberaba Welisson Marques, Otaviano Pereira, André Lemos, Geraldo Lima, Hugo Rufino, Paula Nakamoto e Adriano Martins pelo exemplo de educadores que são, por tanta generosidade e humanidade, por compartilharem tanto saber e pelos inúmeros ensinamentos, que com certeza levarei não somente para minha vida profissional, mas também para minha vida pessoal. Agradeço também, de certa forma, a todos os servidores do IFTM e colaboradores que trabalham direta e indiretamente para o sucesso do programa.

À professora Drª. Crícia Paixão e ao professor Dr. Adriano Martins pelas ricas contribuições para com meu trabalho e pela disponibilidade em compor as bancas de qualificação e defesa.

Aos colegas do mestrado, Flávia, Karuna, Mikaele, Jeanne, Marina, Roni, Luísa, Gabi, Elia, Juno, Cléo e Ricardo, agradeço muito pelo companheirismo, por termos compartilhado

tantos momentos agradáveis, por terem feito da caminhada muito prazerosa. Nossos encontros com certeza deixarão muitas saudades.

Por fim, agradeço ao Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas – Campus Passos pelo incentivo à qualificação.

“Eu serei feliz e orgulhoso se passar o resto da minha vida tentando tornar o sonho de Papert em realidade.”

(RESNICK, 2012 apud MORAIS, 2016, p. 31)

RESUMO

À medida em que os hábitos e a cultura da sociedade se alteram, o processo de ensino-aprendizagem deve sofrer mudanças. Nesse contexto, em 2006, surge o termo "Pensamento Computacional", que sugere um conjunto de habilidades que todos devam ter e não somente os cientistas da computação, e reacende a discussão acerca do ensino de conceitos computacionais à população em geral, sendo que, para o desenvolvimento do Pensamento Computacional, o ensino de programação a alunos da Educação Básica tem sido a prática mais adotada. Enquanto a programação já está inserida nos currículos do ensino regular em diversos países, no Brasil as iniciativas são poucas e autores apontam a falta de estudos que visem objetivos pedagógicos e a qualidade do processo de ensino-aprendizagem. Diante disso, o presente trabalho objetivou elaborar uma proposta desplugada e multiparadigmática para o ensino de programação na Educação Básica. Desplugada no sentido em que nenhuma ferramenta computacional se faz necessária e multiparadigmática no sentido em que não prioriza o ensino de características de um único paradigma de programação, além de não objetivar o ensino de uma linguagem ou técnicas de programação específicas. Propõe-se, então, neste trabalho, uma metodologia de ensino embasada nas ideias de Alan Kay com a programação orientada a objetos, na filosofia LOGO de Seymour Papert, na robótica situada, na computação desplugada e na programação multiparadigmática. Para isso, fez-se um estudo para se obter uma visão geral do que tem sido feito acerca da temática no Brasil e no mundo e fez-se um estudo para se obter um aprofundamento nos assuntos centrais do trabalho. Por fim, neste trabalho, salienta-se que o ensino de programação à alunos da Educação Básica não deva ter como objetivo o ensino de linguagens ou técnicas de programação, mas sim de fornecer materiais concretos a esses alunos para que eles construam suas próprias estruturas do conhecimento. A programação como meio para outras aprendizagens.

Palavras-chave: Ensino de programação. Educação Básica. Pensamento Computacional.

ABSTRACT

As the society habits and culture change, the teaching and learning process must be changed. In this context, in 2006, the term “Computational Thinking” emerge, that suggests a skill set that everyone should have and not just computer scientists, and rekindles the discussion about the teaching of computational concepts to the general population, and for the development of Computational Thinking, the teaching of programming to students of regular education has been the most adopted practice. While programming is already inserted in the regular education curriculum in several countries, in Brazil there are few initiatives and authors point out the lack of studies aimed at pedagogical objectives and the quality of the teaching-learning process. Given this, the present work aimed to elaborate an unplugged and multiparadigmatic proposal for the teaching of programming in regular education. Unplugged in the sense that no computational tools are necessary and multiparadigmatic in the sense that it does not prioritize the teaching of a single programming paradigm characteristics, nor does it aim to teach a specific programming language or techniques. Therefore, this work proposes a teaching methodology based on Alan Kay's ideas with object-oriented programming, Seymour Papert's LOGO philosophy, situated robotics, unplugged computing and multiparadigmatic programming. For this, a study was made to obtain an overview of what has been done about the theme in Brazil and in the world, and a study was made to obtain a deepening in the core subjects of the work. Finally, in this work, it is emphasized that the teaching programming to regular education students should not be aimed at teaching specific programming languages or programming techniques, but rather to provide concrete materials for these students to build their own knowledge structures. The programming as a way for other learnings.

Keywords: Programming teaching/programming learning. Regular education. Computational Thinking.

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

O processo de ensino-aprendizagem deve sofrer mudanças à medida em que os hábitos e a cultura populares se alteram, de forma a se manter contextualizado, motivador e enriquecedor. Diante disso, emergiu a discussão acerca da inserção da Informática na escola e, em 1997, foi criado o Programa Nacional de Informática na Educação (ProInfo), atualmente conhecido como Programa Nacional de Tecnologia Educacional. Com o programa, muitas escolas passaram a ter em seus currículos a disciplina "Introdução à Informática", que visa propiciar aos alunos o contato com as tecnologias digitais. Porém, essa não é mais uma temática da atualidade, visto que, com o avanço dessas tecnologias, principalmente com a popularização dos smartphones, as crianças em geral passaram a ter contato direto com tais tecnologias desde os seus primeiros anos de vida. Além disso, a proposta do ProInfo visa somente o uso das tecnologias e está longe de propiciar aos alunos o desenvolvimento de habilidades computacionais que vêm sendo fortemente discutidas nos âmbitos acadêmico, científico e mercadológico (MOTA et al, 2014).

O desenvolvimento dessas habilidades ganhou atenção em 2006, com o surgimento do termo "Pensamento Computacional", proposto por Wing (2006). O Pensamento Computacional trata-se, em linhas gerais, de habilidades que qualquer pessoa em pleno século XXI deva ter, e não somente cientistas da Computação, como o raciocínio lógico e a abstração e formalização de problemas (BORDINI et al, 2017). O artigo de Wing, "*Computational thinking*" (WING, 2006), gerou uma enorme discussão acerca do ensino de conceitos computacionais no ensino básico, dentre eles, a programação. Iniciou-se, então, um grande movimento para a disseminação do Pensamento Computacional (ARAÚJO; ANDRADE; GUERRERO, 2016; SANTOS; ARAUJO; BITTENCOURT, 2018). Porém, embora muitos estudos que envolvem o ensino de programação no ensino básico tenham sido realizados mundialmente, destaca-se, no Brasil, a falta de estudos que visem a qualidade e profundidade do processo de ensino-aprendizagem, que visem objetivos pedagógicos a fim de se obter aprendizagens mais significativas (SANTOS; ARAUJO; BITTENCOURT, 2018).

Embora a temática tenha ganhado popularidade apenas durante a última década, ela não é tão recente. Na década de 60, Seymour Papert, embasado nos trabalhos de Piaget, propôs um ambiente computacional para aprendizagem de crianças por meio da programação. E, na década de 70, Alan Kay, após contato com os trabalhos de Papert e Piaget com as crianças, trabalhou na criação do Dynabook, precursor da computação móvel, que tratava-se

de uma máquina que proporcionava a crianças uma aprendizagem diferente da habitual. Na primeira versão do Dynabook de Alan Kay era utilizada somente a linguagem Smalltalk, também idealizada por ele, tida como a primeira linguagem puramente orientada a objetos. Naquela época, os trabalhos de Papert e Kay com as crianças não ganharam tanta visibilidade, uma vez que os computadores ainda não eram de uso da população em geral. Com o reacender da discussão em torno do contato com conceitos computacionais por parte de alunos do ensino básico, tais trabalhos têm sido resgatados e embasado muitas pesquisas, visto as inúmeras contribuições que tais autores trazem.

Dentre as práticas adotadas para disseminação do Pensamento Computacional, a programação tem se destacado. E, dentre as justificativas apontadas para o ensino de programação no ensino básico, destaca-se a construção do pensamento crítico e o desenvolvimento de habilidades como o raciocínio lógico, a capacidade de abstração e a formalização de problemas, habilidades essas exigidas a qualquer profissional do século XXI. Algumas dessas habilidades, inclusive, auxiliam os alunos no entendimento de outras disciplinas, como a Matemática e a Física. Destaca-se também a compreensão de como as tecnologias são construídas, a desmistificação do que é a Computação e o incentivo ao desenvolvimento de novas tecnologias (ZANATTA, 2015; OLIVEIRA et al, 2014; PEREIRA JÚNIOR et al, 2005; SILVA, NASCIMENTO, 2012).

Diversas abordagens, diferentemente daquelas utilizadas em cursos da área, têm sido utilizadas para o ensino de programação a alunos da Educação Básica, visando, principalmente, tornar o ensino mais atraente e menos embaraçoso. Dentre essas abordagens, destaca-se a computação desplugada, que é inspirada no livro *Computer Science Unplugged* (Ciência da Computação Desplugada, em português) (BELL; WITTEN; FELLOWS, 1998). O livro traz uma série de atividades para o ensino de conceitos computacionais, como números binários, sem o uso do computador. Tal abordagem se mostra interessante, visto que não exige altos investimentos para colocá-la em prática, como o custo de implantação e manutenção de laboratórios de informática, e auxilia na desconstrução da interpretação equivocada de que a Computação se resume ao uso e estudo dos computadores, gerada pela incompreensão ou desconhecimento do que realmente se trata a Computação (BORDINI et al, 2016).

Um dos grandes desafios, se tratando do ensino de programação, está na tentativa de minimizar as dificuldades apresentadas pelos alunos no processo de ensino-aprendizagem. Fato disso são os altos índices de reprovações nas disciplinas de programação de cursos da área. Diante disso, educadores frequentemente discutem qual linguagem e paradigma adotar

em disciplinas introdutórias de programação, visando, principalmente, a amenização deste problema. A escolha do paradigma e linguagem interferem diretamente no processo de ensino-aprendizagem, uma vez que influenciam a maneira ao qual o programador deverá pensar e estruturar a solução para o problema a ser resolvido. A abordagem mais comumente utilizada no ensino é a adoção primeiro do paradigma imperativo e de uma linguagem que segue tal paradigma. O paradigma imperativo é uma abstração da arquitetura de Von Neumann e, nele, estão presentes características como o uso de instruções sequenciais para realizar uma tarefa, que se dá por meio de mudanças de estado através da manipulação de variáveis – abstrações de células de memória que contém dados.

Na abordagem em que se adota o paradigma imperativo primeiro, os alunos, em geral, passam os primeiros semestres em contato somente com esse paradigma para posteriormente terem contato com outros paradigmas e linguagens, sendo que o contato com os principais paradigmas de programação é de extrema importância para a formação dos alunos. Ademais, autores apontam que abordar um único paradigma tem um efeito negativo na competência do programador e na qualidade dos programas. Além das dificuldades apresentadas pelos alunos logo de início, estar habituado a um paradigma e aprender outro é uma tarefa considerada árdua (BERGIN, 2000 apud VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008; DECLUE, 1996 apud VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008; GUZDIAL, 1995 apud VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008; LATTANZI; HENRY, 1996 apud VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008). Ora, aprender uma linguagem que segue o paradigma imperativo não se trata meramente de aprender uma língua nova, requer conhecimentos técnicos computacionais acerca de hardware, isso também contribui para a dificuldade dos alunos e vai contra o que se deseja em ensinar programação para alunos do ensino básico.

Nesse sentido, tem-se discutido o uso de linguagens multiparadigma para o ensino de programação. Até meados da década de 80, cada linguagem de programação seguia um único paradigma de programação e as linguagens multiparadigma surgiram da necessidade de se construir a solução para um problema utilizando-se técnicas de diferentes paradigmas. Linguagens multiparadigma são, portanto, linguagens que possuem características de dois ou mais paradigmas de programação, dando, assim, ao programador uma maior flexibilidade na escolha do paradigma mais adequado para a resolução de determinado problema. A presença de diversas linguagens multiparadigma na atualidade reforça a necessidade de se abordar os diversos paradigmas e de se buscar uma forma diferente da que tem sido utilizada para o ensino da programação.

Diante disso, o presente trabalho tem como principal objetivo a elaboração de uma proposta multiparadigmática e desplugada para o ensino de programação na Educação Básica. Para alcançar esse objetivo, primeiramente, realizou-se um estudo bibliográfico para se ter uma dimensão dos trabalhos que têm sido realizados no Brasil e no mundo acerca da temática. Posteriormente, fez-se um estudo acerca da presença dessa visão multiparadigmática na programação de computadores e no ensino de programação. Depois disso, foi necessário aprofundamento em temáticas que serviriam como base para a abordagem a se propor. Por fim, idealizou-se e descreveu-se uma metodologia de ensino de programação que pode ser utilizada para aprendizagem de crianças e adolescentes, que se dá por meio de um jogo *desplugado*.

Neste sentido, o Capítulo 1, intitulado *O ensino da programação na Educação Básica e a presença de uma visão multiparadigmática no ensino da programação*, irá trazer uma visão sobre o que tem sido feito e discutido no âmbito do ensino da programação a alunos da Educação Básica e dissertará acerca da presença de uma visão multiparadigmática na programação de computadores e no ensino de programação.

A metodologia de ensino proposta neste trabalho tem como bases: i. a robótica situada, no sentido que a programação não é pré-dada, mas, ao invés disso, ela emerge da relação entre objeto e ambiente ao qual ele está inserido; ii. a programação orientada a objetos proposta por Alan Kay, em que um programa é composto por objetos e que é possível interagir com esses objetos por meio do envio de mensagens, e, talvez o mais relevante, a ruptura com a dependência conceitual entre programa e máquina; iii. a filosofia LOGO, de Seymour Papert, no sentido que o objetivo da metodologia não se trata do ensino de uma linguagem ou técnica de programação específica, mas sim de fornecer materiais concretos aos alunos para a construção por eles mesmos de suas próprias estruturas do conhecimento. No caso, os materiais concretos são fruto da programação. O Capítulo 2, *A robótica situada, a POO de Alan Kay, a filosofia LOGO e a programação multiparadigmática: "novos" olhares para o ensino da programação na Educação Básica*, discorrerá sobre tais ideias, que, novamente, embasam a metodologia proposta.

Por fim, o Capítulo 3, *Metodologia para ensino de programação na Educação Básica: uma proposta*, descreverá e discorrerá sobre a metodologia de ensino de programação que aqui se propõe.

CAPÍTULO 1
O ENSINO DA PROGRAMAÇÃO NA EDUCAÇÃO BÁSICA E A PRESENÇA DE UMA VISÃO MULTIPARADIGMÁTICA NO ENSINO DE PROGRAMAÇÃO

À medida em que a cultura e os hábitos da população se alteram, o processo de ensino-aprendizagem também deve sofrer mudanças a fim de tornar este processo contextualizado, enriquecedor e motivador. Com a globalização e a inserção dos computadores no cotidiano dos cidadãos, se fez necessária a discussão acerca da utilização da tecnologia como ferramenta de apoio no processo de ensino-aprendizagem e inserção da informática na escola. De acordo com o conceito desenvolvido por Jonassen (2007 apud FERRI; ROSA, 2016), os computadores podem ser vistos como "ferramentas cognitivas", uma vez que auxiliam na construção do conhecimento e incentivam o pensamento crítico, criativo e reflexivo, habilidades fundamentais para os dias de hoje de acordo com a Unesco (2009 apud FERRI; ROSA, 2016).

Diante disso, em 1997, foi criado pelo Ministério da Educação (MEC), o Programa Nacional de Informática na Educação (ProInfo), que se destina a promover o uso das tecnologias como ferramenta de apoio pedagógico. Em 2007, o ProInfo se tornou Programa Nacional de Tecnologia Educacional[1] e a disciplina "Introdução à Informática" passou a ser inserida nos currículos da educação básica. As escolas interessadas deveriam se cadastrar no programa e recebiam apoio financeiro para a instalação de infraestrutura adequada para viabilizar o ensino de Informática à crianças e adolescentes. Dessa forma, os alunos em geral passaram a ter contato direto com a Informática, mesmo aqueles que nunca tiveram acesso a um computador, uma vez que este era o principal objetivo do ProInfo.

Porém, diante disso, a iniciativa visa somente o uso das tecnologias pelos alunos e está longe de propiciar a esses alunos adquirir conhecimentos e habilidades para a construção de novas tecnologias, ou de fazê-los compreender o funcionamento de tais tecnologias, ou prepará-los para o ingresso em cursos da área da computação (MOTA et al, 2014). À época em que foi criado, o programa realmente tratava de uma necessidade da sociedade na ocasião: propiciar o acesso às tecnologias digitais. Entretanto, com o "boom" da Internet na década de 90 e, principalmente, a popularização dos smartphones nos anos 2000, essa deixou de ser uma necessidade da população em geral. As crianças passaram a ter contato direto com as tecnologias digitais desde os primeiros anos de vida e, muitas das vezes, possuem um conhecimento muito mais abrangente em relação a essas tecnologias do que o próprio professor. Ademais, o custo para instalação e manutenção de um laboratório de informática é

[1] http://www.fnde.gov.br/index.php/programas/proinfo

bastante elevado, requer recursos financeiros para compra de materiais, requer o trabalho de um técnico de laboratório, além dos gastos com energia elétrica e serviços de Internet. Outra questão que envolve orçamento está relacionada ao fato de que as tecnologias computacionais se tornam obsoletas rapidamente, evoluem de forma rápida. Dessa forma, manter um laboratório de informática funcional requer também que o mesmo seja constantemente atualizado. O ProInfo, portanto, caso queira prosperar, deve atualizar seus objetivos e pensar em alternativas que rompam essa barreira econômica.

A inserção da Informática na educação a partir do uso de ferramentas de apoio pedagógico, portanto, não atende a necessidades consideradas essenciais a profissionais do século XXI. Neste contexto, em um artigo publicado em 2006, intitulado "*Computational thinking*" (WING, 2006), Wing apresenta o termo "Pensamento Computacional", que, de maneira ampla, trata-se de um conjunto de técnicas e habilidades que qualquer pessoa em pleno século XXI deva conhecer/desenvolver, e não somente cientistas da Computação (BORDINI et al, 2017). O termo foi posteriormente definido de maneira operacional pela *Computer Science Teachers Association* (CSTA), pela *International Society for Technology in Education* e por colaboradores.

Ao longo de sua formação, os cientistas da computação desenvolvem, por exemplo, a capacidade de resolver problemas complexos utilizando diversos níveis de abstração e raciocínio lógico. Neste sentido, o Pensamento Computacional pode ser compreendido como o pensar de maneira abstrata, a decomposição de atividades complexas e o uso de avaliação e generalização para resolução de problemas (SELBY; WOOLLARD, 2012 apud ARAÚJO; ANDRADE; GUERRERO, 2016). A CSTA (2011 apud SANTOS et al, 2015) define ainda que o processo relacionado ao Pensamento Computacional inclui, entre outras características, a formalização de problemas, a organização e análise lógica de dados, a representação de dado por meio da abstração e a generalização da solução de um problema para que possa ser utilizada na resolução de outros problemas.

Wing (2006) salienta que o Pensamento Computacional "é uma habilidade fundamental para qualquer um e não somente para cientistas da computação" (WING, 2006, p. 33), envolve não somente a resolução de problemas, mas também o entendimento do comportamento humano. Para Wing (2006), a computação possui bases sólidas para, ao tratar da resolução de um problema, responder questões, como "Qual é a dificuldade de resolver?" Ou "qual é a melhor maneira de resolver isso?" (WING, 2006, p. 33). O pensamento computacional se faz presente, por exemplo, ao decompor problemas complexos em partes menores, mais fáceis de serem resolvidas. Wing (2006) salienta ainda diversas situações em

que o pensamento computacional está presente no cotidiano das pessoas, mas acredita que o pensamento computacional somente estará enraizado na sociedade quando palavras como algoritmo fizeram parte do vocabulário da população em geral. Wing elenca ainda uma série de características do pensamento computacional, dentre elas a de que pensar como um cientista da computação não se trata meramente de saber programar um computador, mas é preciso, além disso, pensar em diversos níveis de abstração. Outra característica é que, apesar de as bases da computação estarem sobre a matemática, o pensamento computacional se difere do pensamento matemático em sua abrangência. Também, não são os artefatos gerados pela computação que terão relevância, mas sim todos os conceitos e ideias computacionais que foram utilizados para gerar tais artefatos.

O artigo de Wing (2006) e o posterior desenvolvimento do termo "Pensamento Computacional" geraram uma enorme discussão, mundialmente, na comunidade acadêmica e científica acerca da educação em Computação, mais especificamente acerca das razões pelas quais a Ciência da Computação deva ser abordada no ensino básico e deva ser considerada uma ciência. A Ciência da Computação é então equiparada a ciências como Matemática e Biologia, e as habilidades relacionadas ao Pensamento Computacional são equiparadas às habilidades de ler e escrever (ARAÚJO; ANDRADE; GUERRERO, 2016).

Desde o surgimento do termo Pensamento Computacional, um intenso movimento visando sua difusão foi iniciado, tanto na academia, quanto fora dela. Diversos trabalhos visando o ensino de conceitos da Computação para alunos do ensino básico foram e têm sido desenvolvidos. De acordo com o trabalho de Santos, Araújo e Bittencourt (2018), 2010 foi o ano em que se teve um crescimento mais significativo no número de pesquisas relacionadas ao tema, tanto no Brasil, quanto no mundo, com uma publicação de 19 trabalhos, se comparado com somente 8 publicações no ano anterior e com um total de 14 publicações entre os anos de 2001 e 2008. Os autores destacam também que esse número vem crescendo a cada ano, sendo 22 trabalhos publicados em 2012, 41 em 2013, 86 em 2015 e 81 em 2016. E, embora o Pensamento Computacional tenha sido trabalhado de diversas maneiras, como através da robótica, destaca-se que a programação, seguida da computação desplugada, têm sido as abordagens mais utilizadas (SANTOS; ARAUJO; BITTENCOURT, 2018).

A computação desplugada é inspirada no livro *Computer Science Unplugged*[2] e visa o ensino de conceitos computacionais sem o uso de um computador (BELL; WITTEN; FELLOWS, 1998). *Computer Science Unplugged* trata-se de um livro para o ensino de

[2] http://csunplugged.org/books/

conceitos computacionais a crianças de forma lúdica e sem o uso do computador. O livro traz uma série de atividades que trabalham o entendimento de conceitos fundamentais da computação, como números binários. Todas as atividades possuem respostas e, ao final de cada uma delas, uma descrição do motivo pelo qual a atividade foi trabalhada. O livro foi escrito por professores de Ciência da Computação e professores que atuam em escolas da educação básica, e possui tradução para diversas línguas, inclusive para o Português (LENZ; CAMBRAIA, 2015). Um dos objetivos da computação desplugada é eliminar as barreiras técnicas, como a falta de disponibilidade de computadores e acesso à Internet, e as interpretações equivocadas do que realmente é a Computação, como a de que a Computação se resume ao estudo dos computadores (BORDINI et al, 2016).

A Computação muitas vezes é vista como a área que trata dos computadores, porém a Computação surgiu muito antes dos primeiros computadores. Wazlawick (2017) divide a história da Computação em 11 eras, que vão desde o surgimento das primeiras formas de contar, que compreende até os anos de 1622, até o surgimento da computação móvel, nos anos 2000. A computação surgiu da necessidade de se realizar cálculos cada vez mais complexos, decorrente do avanço da astronomia e da engenharia. Tem-se, portanto, entre os séculos XVII e XIX, o surgimento das primeiras calculadoras mecânicas. Os primeiros computadores começaram a surgir somente a partir do ano de 1936, embora antes disso tenham ocorrido diversos acontecimentos que marcam a história da Computação e possibilitam o surgimento dos computadores. A partir disso, a Computação passa a absorver diversas outras características e responsabilidades, como o processamento de informações. Nesse sentido, a Computação possui diversas teorias de si que são amplamente aceitas, como a de que a Computação trata do processamento de informações, ou que a Computação trata da computabilidade efetiva. Porém, nenhuma delas a explica em toda a sua abrangência. O fato é que a Computação ainda não possui uma teoria única que a defina como um todo e, por isso, essa é ainda uma questão bastante ampla e em aberto. (WAZLAWICK, 2017; SMITH, 2002).

Retomando à questão do desenvolvimento do Pensamento Computacional a partir do ensino de programação, apesar do tema ter ganhado destaque somente na última década, não se trata de um assunto recente. Os primeiros passos para o contato de usuários comuns com a programação se deram na década de 60 com a proposta de Papert para o ensino de matemática através da programação *Logo* (LYE; KOH, 2014), um *framework* onde os alunos devem mover a tartaruga, uma personagem do *Logo*, através de comandos baseados em linguagem comum, como o comando “parafrente” para mover a tartaruga para frente (ZANATTA, 2015). Entretanto, devido a não familiaridade das pessoas com tecnologias naquela época, a prática

não obteve muita repercussão. Santos, Araújo e Bittencourt (2018), citam também a criação de um currículo modelo para o Ensino Médio em Israel na década de 90.

Uma das justificativas para se ensinar programação, principalmente para alunos da educação básica, é a grande demanda por profissionais na área e, ao mesmo tempo, a falta desses no mercado de trabalho. Como fatores agravantes, pesquisas apontam um aumento dessa demanda e uma diminuição da procura por cursos da área (CABRAL, 2007 apud GARCIA; CORREIA; SHIMABUKURU, 2008; BARROS, 2011 apud RODRIGUES, 2013), além de um alto índice de evasão e reprovação, principalmente em disciplinas relacionadas à programação (CABRAL, 2007 apud MARQUES et al, 2011; BARROS, 2011 apud RODRIGUES, 2013). Estas evasões e reprovações vêm sendo relacionadas à dificuldade dos alunos em Ciências Exatas, como formalizar problemas, fazer abstrações e situações que envolvem o raciocínio lógico, e não somente em cursos da área da Ciência da Computação, mas também em vários outros cursos que exigem um forte embasamento matemático (PEREIRA JÚNIOR; RAPKIEWICZ, 2004 apud GARCIA; CORREIA; SHIMABUKURU, 2008).

Outro fator citado como contribuinte para as altas taxas de evasão e reprovação se refere à defasagem dos alunos em relação a conteúdos do ensino regular, como conteúdos básicos de exatas. Muitas vezes, alunos chegam à graduação sem saber efetuar operações básicas matemáticas. É apontada também a falta de habilidades que deveriam ter sido adquiridas na Educação Básica, como o pensamento crítico e estruturado. (BARBORA, 2011 apud OLIVEIRA et al, 2014; PRADO, 2000 apud ANACLETO; BOENO; ALBERTON, 2012). O pensamento estruturado deve ser construído de maneira progressiva, uma vez que, com o passar dos anos, os alunos se deparam com problemas de níveis cada vez mais complexos que exigem o raciocínio lógico cada vez mais incrementado (PIAGET, 1975 apud PEREIRA; BEZERRA JÚNIOR, 2014). Para Piaget, o desenvolvimento da inteligência se dá por um processo de construção de estruturas do conhecimento (PIAGET, 1967 apud FERRACIOLI, 1999). Cada “conhecimento novo” se interage com uma estrutura já existente, havendo um processo de reestruturação e criação de uma nova estrutura do conhecimento, desenvolvendo-se, assim, a mente do indivíduo (PIAGET; LNHELDER, 1978 apud FERRACIOLI, 1999). O pensamento estruturado ocorre por meio do processo de assimilação e acomodação (PIAGET, 1982 apud FERRACIOLI, 1999; PIAGET, 1971 apud VARELA; BARBOSA, 2007) e é obtido através da aprendizagem da lógica. (PIAGET, 1975 apud PEREIRA; BEZERRA JÚNIOR, 2014). Dessa forma, é importante que o raciocínio lógico seja trabalhado desde os primeiros anos do ensino fundamental (PEREIRA; BEZERRA

JÚNIOR, 2014). E, em contrapartida, a programação auxilia no processo de construção do pensamento crítico e no desenvolvimento cognitivo (ZANATTA, 2015; SILVA; NASCIMENTO, 2012). Para Steve Jobs, “programar ensina a pensar” (ZANATTA, 2015; OLIVEIRA et al, 2014).

Além disso, alguns autores afirmam ser importante, em pleno século XXI, pessoas adquirirem conhecimento computacional e não apenas fazer uso das tecnologias desenvolvidas. (CODE CLUBE BRASIL, 2015 apud ZANATTA, 2015; OLIVEIRA et al, 2014). Neste sentido, outra justificativa apontada para o ensino de programação é quanto ao incentivo à criação de novas tecnologias. Sachs (2000 apud PEREIRA JÚNIOR et al, 2005) destaca a importância do incentivo à criação de novas tecnologias: os países podem ser divididos em três tipos – os que detém tecnologia, os que absorvem tecnologia e os que não têm acesso à tecnologia, sendo, geralmente, os primeiros, países desenvolvidos, os segundos, países subdesenvolvidos, e, por último, países emergentes, aqueles que não têm acesso a tecnologia, ficando mais susceptíveis à pobreza.

Para Oliveira et. al. (2014), não ensinar fundamentos da computação desde cedo resulta em uma falta de interesse por parte dos alunos em seguir a área e em um não conhecimento do tema. Os autores destacam ainda que a prática é benéfica para todos os alunos e não apenas para aqueles queiram iniciar em cursos voltados para a informática, pois proporciona o desenvolvimento de habilidades muitas vezes não adquiridas através do ensino regular. Ademais, contribui para o entendimento de outras matérias, principalmente as das ciências exatas, como Física e Matemática.

Apesar dos inúmeros incentivos ao ensino de programação fora do contexto de cursos específicos da área, há aqueles que são contrários à prática. Dentre as justificativas, está a de que programação é algo especializado, específico, e não algo básico como ler e escrever, por isso não é esperado que todos saibam (ZANATTA, 2015). Neste sentido, cabe destacar que, tomando as ideias de Alan Kay e Seymour Papert, que serão abordadas no Capítulo 2, uma abordagem possível não se trata do ensino de uma linguagem de programação e/ou técnicas de programação específicas, mas sim de propiciar aos alunos adquirir habilidades que serão exigidas deles futuramente, seja enquanto alunos de graduação ou enquanto profissionais de qualquer área, através de atividades que lidam com fundamentos essenciais à programação, como a resolução de problemas, o raciocínio estruturado e lógico, etc.

Outra justificativa apontada para o não ensino de computação a alunos da Educação Básica é a de que aprender a programar pode prejudicar crianças e adolescentes em seu desenvolvimento, principalmente no que diz respeito ao convívio social, e também pode

privá-los de experiências próprias de cada faixa etária, como brincar com outras crianças (ZANATTA, 2015). De fato, o uso excessivo e inadequado das tecnologias digitais podem ser prejudiciais às crianças e aos adolescentes, porém é fato que tais tecnologias estão presentes no dia a dia da população em geral e a prática que aqui se discute pode inclusive vir a contribuir para essa questão, mostrando aos alunos um uso mais adequado e produtivo de tais tecnologias, um uso que visa contribuir para o desenvolvimento das crianças e dos adolescente.

1.1 O ensino de programação no mundo

Nos Estados Unidos, no ano 2000, foi publicada a primeira versão do *Model Curriculum for K-12 Computer Science*, que fornece diretrizes para a inserção da Ciência da Computação nos currículos do ensino básico americano. O currículo modelo traça linhas para que educadores sigam a fim de garantir que alunos adquiram habilidades necessárias diante de um mundo globalmente competitivo e tecnológico. Trata-se de um guia que propõe o ensino de conceitos computacionais de forma estruturada a alunos da chamada K-12, que compreende as 12 séries do ensino básico americano. Estes conceitos são considerados básicos para que um cidadão adquira habilidades imprescindíveis para a obtenção de sucesso nos dias atuais (TUCKER, 2003).

Em 2013, diante da falta de profissionais no mercado na área da Tecnologia da Informação e da crescente demanda por estes profissionais, dois irmãos, Ali e Hadi Partovi, lançaram o projeto Code.org[3], uma Organização Não Governamental (ONG), que visa incentivar o ensino de programação para todos, principalmente para crianças, jovens e adolescentes. A iniciativa teve inclusive o apoio de grandes nomes da área tecnológica, como Bill Gates, Mark Zuckerberg e Jack Dorsey, fundadores, respectivamente, da Microsoft, Facebook e Twitter. Para eles, o conhecimento em programação se faz importante e está sendo cada vez mais exigido (SILVA et al, 2015). Outras iniciativas mundialmente lançadas também podem ser destacadas, como a "Hora do Código" (*Code Hour*)[4], o CODE Club[5] e a ferramenta Scratch[6].

A "Hora do Código" é um movimento criado pelo Code.org que atinge milhões de estudantes em vários países. A ideia é que qualquer um, em qualquer lugar do mundo, possa

[3] https://code.org/
[4] https://hourofcode.com/br
[5] https://www.codeclubworld.org/
[6] https://scratch.mit.edu/

organizar uma hora do código, através de tutoriais disponíveis na internet em vários idiomas. Podem participar pessoas a partir de 4 anos de idade (ZANATTA, 2015).

O CODE Club (Clube do Código, em português), ou CODE Club World, é uma rede mundial com o objetivo de promover o ensino de programação de computadores a crianças. A rede é totalmente gerenciada por voluntários e surgiu em 2012, a partir de um projeto voluntário desenvolvido por Clare Sutcliffe e Linda Sandivik, no Reino Unido. A ideia do CODE Club é que clubes sejam criados no mundo todo por voluntários, com objetivo de passar adiante conhecimentos acerca de programação através de cursos.

Já Scratch é uma ferramenta online para programação voltada principalmente para crianças e adolescentes, mas que qualquer um pode usar. Jogos, por exemplo, podem ser criados de maneira lúdica, apenas arrastando blocos, sem ser preciso digitar uma linha de código. A ferramenta Scratch é inspirada no ambiente de programação LOGO, proposto por Papert na década de 60.

Outras iniciativas mundiais são citadas pelos autores. A Inglaterra teve seu currículo do ensino básico alterado em 2014 e as crianças passaram a aprender computação (BERRY, 2013 apud FERRI; ROSA, 2016). No Japão, os alunos aprendem desde as séries iniciais programação na disciplina de Matemática (FERRI; ROSA, 2016). E, no Reino Unido, Nova Zelândia, Alemanha, Índia e Coreia do Sul, a Computação já está presente nos currículos dos ensinos correspondentes ao Ensino Médio (ROYAL SOCIETY, 2012, apud SANTOS; ARAUJO; BITTENCOURT, 2018).

O ensino de programação para crianças já é, portanto, um assunto avançado em diversos outros países, como nos Estados Unidos, China e Estônia, que adotaram o ensino de programação desde cedo (SILVA et al., 2015) (ZANATTA, 2015) (OLIVEIRA et. al., 2014).

1.2 O ensino de programação no Brasil

Em 2004, a Sociedade Brasileira de Computação (SBC) propôs a inserção de conceitos computacionais, como a programação, no Ensino Médio (PEREIRA JÚNIOR et al, 2005). Desde então, pesquisas que investigam os benefícios para tal prática foram desenvolvidas.

Em 2013, eram apontadas escolas que inseriram em seus currículos o ensino da programação a alunos da educação básica, como as escolas Móbile e Vértice, ambas de São Paulo (HONORATO, 2013; PEREIRA, 2013). Estas iniciativas ainda resistem e novas surgem, porém crescem de maneira lenta (OLIVEIRA et al, 2014), mesmo diante de tantas

outras iniciativas que visam incentivar pessoas em geral a aprender programação, como o CODE Club Brasil e os projetos “Programaê” e o “IAI?”.

O CODE Club Brasil[7] é uma iniciativa brasileira para ensino de programação a crianças a partir de 9 anos utilizando a metodologia proposta pelo CODE Club. O Programaê[8] é um projeto que visa auxiliar professores no ensino de programação, apresentando-lhes diversos planos de ensino baseados no Code.org e na ferramenta Scratch, que foram apresentados na seção anterior. E o “IAI?”[9] trata-se de uma escola de São Paulo que ensina a crianças o uso de várias tecnologias, como programação e robótica (ZANATTA, 2015).

Recentemente, a SBC tem trabalhado mais efetivamente para que a Computação seja inserida nos currículos da Educação Básica, cabendo destacar a criação, em 2017, dos “Referenciais de Formação em Computação: Educação Básica” (SBC, 2017, apud SANTOS; ARAUJO; BITTENCOURT, 2018).

Porém, Santos, Araújo e Bittencourt salientam que, enquanto países como Estados Unidos e Nova Zelândia tentam consolidar suas práticas educativas, os trabalhos desenvolvidos no Brasil objetivam facilitar a inserção da computação na Educação Básica através da tentativa de envolver os alunos com a Computação e mostrar a eles os benefícios disso. Faltam estudos que visem a qualidade e profundidade do processo de ensino-aprendizagem, que visem objetivos pedagógicos a fim de se obter aprendizagens significativas. Para os autores, “(...) há uma preocupação intensa em como a computação é apresentada para atrair a atenção dos estudantes, porém há pouca preocupação com se a prática utilizada é efetiva para a aprendizagem.” (SANTOS; ARAUJO; BITTENCOURT, 2018, p. 7).

No mais, as motivações para os estudos realizados no Brasil são diversas e incluem atrair os alunos para cursos da área de Computação, incentivar a criação de tecnologias, diminuição da evasão dos cursos da área de Computação e desenvolver o Pensamento Computacional (BORDINI et al, 2016).

1.3 O ensino de programação e a presença de uma visão multiparadigmática

De acordo com as Diretrizes Curriculares de Cursos da Área de Computação e Informática, o principal objeto do ensino de programação está em tornar o aluno capaz de aplicar os conhecimentos adquiridos em sua formação na elaboração de uma solução

[7] http://www.codeclubbrasil.org.br/
[8] http://programae.org.br/
[9] http://site.iai.art.br/

computacional a um problema, ou seja, “o importante para o aluno é saber aplicar de maneira coerente as técnicas e abordagens estudadas através dos paradigmas de programação.” (BLATT; BECKER; FERREIRA, 2017, p. 816).

O ensino de programação em cursos da área de Computação se dá comumente através de aulas teóricas tradicionais e posterior aplicação, em laboratório de informática, dos conceitos abordados, utilizando-se uma linguagem de programação. Embora esta seja uma etapa importante para o processo de ensino-aprendizagem de programação, não deve ser utilizada unicamente, uma vez que pode acarretar em um processo de somente reprodução daquilo que foi ensinado. Em paralelo a isso, as disciplinas de programação são as que apresentam um maior número de reprovações (BLATT; BECKER; FERREIRA, 2017). Diversos estudos apontam altos índices de reprovação, principalmente em disciplinas relacionadas à programação (CABRAL, 2007 apud MARQUES et al, 2011; BARROS, 2011 apud RODRIGUES, 2013), e evasão, além de uma diminuição da procura por esses cursos (CABRAL, 2007 apud GARCIA; CORREIA; SHIMABUKURU, 2008; BARROS, 2011 apud RODRIGUES, 2013). Os estudos relatam, ainda, que essas evasões e reprovações estão relacionadas às dificuldades dos alunos em questões como formalizar problemas, fazer abstrações e em situações que envolvem o raciocínio lógico (PEREIRA JÚNIOR; RAPKIEWICZ, 2004 apud GARCIA; CORREIA; SHIMABUKURU, 2008). A não familiaridade com um determinado paradigma de programação é também apontado como um contribuinte para as dificuldades encontradas pelos alunos no processo de ensino-aprendizagem de programação (BLATT; BECKER; FERREIRA, 2017).

Os paradigmas de programação mais populares são o paradigma imperativo, o paradigma orientado a objetos, o paradigma funcional e o paradigma lógico. Diz-se que uma linguagem de programação segue um ou mais paradigmas. Ao criar um programa em uma linguagem imperativa, por exemplo, que segue o paradigma imperativo, o programador estará usando técnicas de programação que consistem em características do paradigma imperativo, ou que permitem a criação de um programa sob o paradigma imperativo. Uma outra linguagem imperativa também terá essas mesmas características.

O conceito de paradigma de programação pode ser entendido sob duas perspectivas: i) perspectiva em que as linguagens ditam os paradigmas – nesta perspectiva, pode-se dizer que os paradigmas são definidos a partir de linguagens e um paradigma é visto como “uma visão abstrata de uma classe de linguagens de programação que descrevem um significado de resolução de problemas de programação” (HAILPERN, 1986a, p. 8); ii) perspectiva em que os paradigmas ditam as linguagens – neste caso, um paradigma trata-se de uma maneira de

enxergar o problema a ser resolvido e as linguagens de programação, por sua vez, são utilizadas para a escrita da resolução de problemas a partir de um ou mais paradigmas. Budd (1995) define paradigma, portanto, como “a maneira (...) como as tarefas que devem ser executadas em um computador devem ser estruturadas e organizadas.” (BUDD, 1995, p. 3).

Observa-se, portanto, que, tanto paradigma, quanto linguagem, influenciam a forma ao qual o programador irá pensar e estruturar a solução para o problema. Diz-se, então, que o paradigma direciona a maneira pela qual o programador enxerga o problema (HAILPERN, 1986a). Os programas escritos em linguagens procedurais, ou procedimentais, por exemplo, tratam-se de um conjunto de procedimentos para a realização de uma tarefa. Logo, para desenvolver uma solução para determinado problema sob esse paradigma, o programador irá pensar nos procedimentos que deverá implementar para a realização de tal tarefa. Já programas escritos em linguagens orientadas a objetos são visto como um conjunto de representações de objetos do mundo real que interagem entre si e podem ter seu estado alterado. Dessa forma, para se criar um programa sob tal paradigma, o programador deverá pensar em quais objetos deverão compor o programa e em como eles deverão se relacionar (SEBESTA, 2011; TUCKER; NOONAN, 2008).

Neste contexto, Budd (1995), afirma que as linguagens de programação, de maneira semelhante às linguagens naturais, são utilizadas pelos indivíduos para comunicar ideias e informações, são usadas para comunicar uma solução computacional a um problema. Assim sendo, Budd (1995) aborda a ideia do linguista americano Edward Sapir, de que “(...) os limites de uma linguagem dirigem (embora não necessariamente limitem) a natureza de um pensamento.” (BUDD, 1995, p. 4), e relaciona essa ideia às linguagens de programação: “(...) a maneira em que um programador pensa que um problema será resolvido irá colorir e alterar, em uma maneira fundamental e básica, o modo em que um algoritmo é desenvolvido.” (BUDD, 1995, p. 4).

A exemplo disso, o autor cita uma situação em que dois programadores implementaram soluções para um mesmo problema, sendo que um deles utilizou a linguagem Fortran e o outro, a linguagem APL. Curiosamente, apesar de a linguagem Fortran ser considerada mais eficiente, o programa criado em APL apresentou maior eficiência. A diferença, neste caso, não estava relacionada à eficiência das linguagens, mas sim às estratégias utilizadas para a resolução do problema. O fato é que o programador Fortran sequer chegou a considerar a solução utilizada pelo programador APL, pois a linguagem Fortran o direcionava a pensar de determinada maneira e não de outra (BUDD, 1995).

A ideia de Sapir é citada em um artigo de Benjamin Whorf (apud BUDD, 1995, p. 4) e é referida como a hipótese Sapir-Whorf. Tal hipótese afirma que “a linguagem em que uma ideia é expressada pode influenciar ou direcionar uma linha de raciocínio” (BUDD, 1995, p. 6). Tal hipótese é confrontada com a conjectura de Church – “qualquer computação para a qual exista um procedimento efetivo pode ser realizada por uma máquina de Turing.” (BUDD, 1995, p. 7). O trabalho de Turing foi fundamental para a fundamentação da Computação. Turing idealizou uma máquina que fosse capaz de realizar cálculos de forma automática através de um conjunto de passos finitos, em que há a manipulação de símbolos presentes em uma fita quadriculada de comprimento ilimitado. Tal máquina é chamada de máquina de Turing e os passos realizados por ela para a realização de um cálculo são chamados de algoritmo (FONSECA FILHO, 2007).

Retomando à conjectura de Church, Budd (1995) salienta, entretanto, que não há uma definição formal para o termo “procedimento efetivo”, o que torna essa conjectura não comprovável ou não comprovada, ao passo que nenhum contra exemplo tenha sido apresentado até então. Mas, considerando a conjectura de Church como verdadeira, pode-se afirmar que qualquer linguagem que simule uma máquina de Turing é “suficientemente poderosa para executar qualquer algoritmo realizável” (BUDD, 1995, p. 7), logo as linguagens de programação são idênticas e uma ideia expressada em uma linguagem pode ser expressada em qualquer outra. Entretanto, o “poder” das linguagens de programação, enquanto habilidade para resolver problemas, é um termo comumente vago. Budd frisa, então, que, tomando como exemplo a habilidade de um nativo de um país de clima quente em descrever a neve e seus diferentes tipos ou usos, a linguagem direciona o pensamento, porém não exclui a possibilidade do pensar sob determinado aspecto. Essa afirmação reforça a ideia de que um programador APL e um programador Fortran podem criar soluções para um mesmo problema, porém as soluções comumente serão distintas, pois cada linguagem direciona o programador a pensar sob determinada perspectiva.

Logo, se não é possível comparar o poder de linguagens de programação, quais critérios utilizar para a escolha de uma linguagem? A resposta está na característica que é considerada pelos programadores como a mais importante: a facilidade de uso. As linguagens não deixam de ser usadas pelo fato de não serem capazes de realizar determinada tarefa, mas por serem difíceis de usar, pelos programas escritos na linguagem ficarem grandes e complexos, ou ficarem um tanto quanto deselegantes. Nas comunidades de programação, frequentemente surgem argumentos de que certa linguagem é mais adequada para determinados tipos de problemas, pois produzem algoritmos melhores (BUDD, 1995).

Essa questão retoma a um assunto frequentemente tratado em estudos sobre conceitos, ou princípios, de linguagens de programação, como na obra de Sebesta (2011) ou de Tucker e Noonan (2008) que é a questão da expressividade de linguagens. Em linguagens de programação, a expressividade é considerada uma característica que afeta diretamente critérios de avaliação de linguagens, que são utilizados justamente para a escolha da linguagem mais adequada para a resolução de determinado problema. A expressividade afeta, por exemplo, a facilidade de escrita, que indica o quão facilmente uma linguagem pode ser usada para a escrita da solução de um problema. Uma linguagem pode ser, portanto, mais adequada para a solução de determinados problemas e outros não. A linguagem C, por exemplo, não se mostra adequada para a construção de interfaces gráficas. É possível, sim, construir interfaces gráficas utilizando a linguagem C, porém é uma tarefa considerada árdua, pois requer a escrita massiva de linhas de código, o que torna os programas grandes, complexos e deselegantes. O mesmo não ocorre com a linguagem Visual Basic, por exemplo, que foi projetada justamente para essa finalidade. Normalmente, os projetistas de linguagens, precisam abrir mão de determinadas características para priorizar outras. Os projetistas da linguagem Java, por exemplo, tiveram de abrir mão de desempenho para garantir portabilidade.

1.4 Linguagens multiparadigma

Até meados da década de 80, cada linguagem seguia um único paradigma de programação e, portanto, ao utilizar determinada linguagem, o programador era obrigado a "pensar" sob determinado paradigma. Hailpern (1986a) cita um tipo de problema frequentemente enfrentado por programadores à época: em meio ao desenvolvimento de um sistema de grande porte, programadores se deparam com uma situação que seria melhor resolvida se estivessem utilizando outra linguagem de programação. Por exemplo, estar utilizando a linguagem Lisp e se deparar com a necessidade de uma complicada manipulação de vetores. Neste caso, o programador possui duas alternativas: continuar utilizando a linguagem Lisp e fazer o melhor uso possível dela, dentro do que a linguagem lhe permite, ou desistir e partir para o uso de outra linguagem, o que muitas vezes não é uma opção considerável, visto que reconstruir um sistema em outra linguagem ou criar módulos do sistema em diferentes linguagens e fazer com que eles se comuniquem são tarefas árduas.

Diante deste cenário, na tentativa da solução de problemas como este, Hailpern (1986a) aponta o início do desenvolvimento de uma nova classe de linguagens: as linguagens

multiparadigma, que não restringem o programador a um único paradigma, mas, sim, combinam dois ou mais paradigmas de programação. Entretanto, Hailpern (1986a) destaca que as linguagens de programação multiparadigma não foram os primeiros sistemas multiparadigma a existirem. Os sistemas operacionais convencionais, por exemplo, podem ser considerados multiparadigma, porém esses sistemas incorporavam muitas linguagens diferentes e requeriam um "editor de ligação que combina arquivos objeto de diferentes compiladores em um único programa executável." (HAILPERN, 1986a, p. 6). Estes editores, por sua vez, eram de difícil utilização e não suportavam qualquer linguagem, além de os programadores terem que dominar cada uma das linguagens (HAILPERN, 1986a).

Pode-se dizer, portanto, que as linguagens multiparadigma surgiram, dentre outras coisas, da necessidade de se construir linguagens que permitissem a criação de um único programa a partir de diferentes paradigmas. Assim, o uso de uma única linguagem de programação evita a necessidade de interfaces de comunicação entre linguagens incompatíveis (MÜLLER; MÜLLER; VAN ROY, 1995). Além disso, as linguagens multiparadigma dão uma maior flexibilidade ao desenvolvedor, uma vez que fornecem um "conjunto 'certo' de construções para o programador, permitindo ao programador usar mais de um modo de pensamento (paradigma) para problemas complexos." (HAILPERN, 1986b, p. 70).

Müller, Müller e Van Roy (1995) definem a programação multiparadigma como sendo "a combinação de vários paradigmas de programação em um modelo simples. De um ponto de vista fundamental, isso permite entender várias formas de computação como facetas de um fenômeno único." (MÜLLER; MÜLLER; VAN ROY, 1995, p. 3). Sendo assim, os programadores podem utilizar diversos estilos de resolução de problemas e selecionar uma técnica que melhor se adeque ao problema a ser resolvido (BUDD, 1995).

Hailpern (1986a) cita que existem pelo menos quatro maneiras de se construir linguagens multiparadigma. Uma delas é a partir da combinação de duas ou mais linguagens. A vantagem disso é quando o programador está familiarizado com pelo menos uma delas. Neste caso, ele começa a utilizar esta linguagem facilmente. A desvantagem é a complexidade da semântica como resultado da interação entre construções sintáticas de linguagens distintas. Outra maneira se dá a partir da adição de novas funcionalidades a uma linguagem existente. (HAILPERN, 1986a). Um exemplo seria a linguagem C++, que adiciona à linguagem C "um conjunto de mecanismos para definição de novos tipos" (HAILPERN, 1986b, p. 71) e suporte a abstração de dados e programação orientada a objetos, além de simplificar algumas

características da linguagem C, como operações para liberação de memória. Ou, ainda, começar do início, criando uma linguagem completamente nova (HAILPERN, 1986a).

Barbosa (2002) menciona que, "atualmente, a comunidade científica dedica consideráveis esforços para a integração de paradigmas básicos" (BARBOSA, 2002, p. 14). E observa-se o surgimento frequente de novas linguagens multiparadigma, sendo que, atualmente, a presença da programação multiparadigma é claramente observada em muitas das linguagens mais recentes, como Python[10], Scala[11], Ruby[12], Groovy[13], F#[14] e tantas outras. Uma das principais motivações para isso está na necessidade de se construir linguagens de fácil uso, característica essa, como já mencionado, considerada a mais importante para a maioria dos programadores. A exemplo disso, estão as linguagens *assembly* (de montagem), que caíram no desuso justamente por serem consideradas de difícil uso (BUDD, 1995).

A facilidade no uso também é almejada para o ensino de programação a alunos da Educação Básica. Nota-se, nos estudos realizados com alunos da Educação Básica, uma significativa preocupação em utilizar abordagens que tornem o ensino de programação mais atrativo e, ao mesmo tempo, menos "embaraçoso e complexo" (SANTOS; ARAUJO; BITTENCOURT, 2018, p. 2). Além disso, o ensino deve ser adequado e coerente ao público em questão. Dessa forma, observa-se o uso de abordagens bastante diferentes das utilizadas para o ensino de programação em cursos da área. Dentre essas abordagens, destaca-se a utilização de desafios para introdução de conceitos básicos e o uso de ambientes visuais de programação (BORDINI et al, 2016). Além disso, há frequentemente a preocupação em propor novos ambientes. Utiliza-se também, com muita frequência, ambientes lúdicos e a programação visual, ou em blocos, como a ferramenta Scratch, a mais comumente utilizada. O uso de ferramentas como essa tem se mostrado positivo para o ensino de programação nas escolas, pois permite que os alunos aprendam a pensar de forma criativa e sistemática na resolução de problemas e a trabalhar de maneira colaborativa (BORDINI et al, 2017; BLATT; BECKER; FERREIRA, 2017). Além disso, é necessário buscar a adequação da técnica e conhecimento ao perfil do aluno.

Diante disso, a programação multiparadigma é vista também como vantajosa para o ensino de programação. Muitos autores acreditam que a programação multiparadigma contribui também para a facilidade no uso, ou seja, torna as linguagens mais fáceis de serem

[10] https://www.python.org/about/
[11] https://docs.scala-lang.org/pt-br/tutorials/tour/tour-of-scala.html.html
[12] https://www.ruby-lang.org/pt/about/
[13] http://groovy-lang.org/
[14] https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/

utilizadas. Müller, Müller e Van Roy (1995) mencionam que a "programação multiparadigma facilita a codificação de algoritmos em um estilo 'natural'." (MÜLLER; MÜLLER; VAN ROY, 1995, p. 3), – esse estilo "natural" seria um estilo mais próximo das linguagens utilizadas pelos seres humanos para se comunicarem – o que, de certa forma, também está relacionado ao nível de abstração da linguagem, que, em outras palavras, se refere ao nível de abstração de detalhes de hardware proporcionado pela linguagem.

1.5 As linguagens multiparadigma e o ensino de programação

A(s) linguagem(ns) e paradigma(s) a serem adotados em disciplinas introdutórias de programação é um assunto frequentemente discutido por educadores diante dos altos índices de reprovações nessas disciplinas, visando, principalmente, amenizar as dificuldades encontradas pelos alunos quanto à aprendizagem de programação (MASON; COOPER, 2014). Em recomendações das duas principais sociedades dos profissionais da Computação – *Association for Computing and Machinary (ACM)* e *IEEE Computer Society* – são citados oito modelos que podem ser adotados para disciplinas introdutórias de programação, sendo eles: abordagem imperativo-primeiro, abordagem orientado-a-objetos-primeiro, abordagem funcional-primeiro, abordagem lógico-primeiro, abordagem hardware-primeiro, abordagem algoritmos-primeiro, abordagem conceitos-primeiro e abordagem amplitude-primeiro.

Os quatro primeiros modelos se referem à adoção de um único paradigma de programação, sendo a abordagem imperativo-primeiro a mais comumente utilizada. Na grande maioria dos casos, os alunos têm um primeiro contato com a programação a partir do paradigma imperativo e de uma linguagem que segue esse paradigma. Os alunos, em geral, passam os primeiros semestres dos cursos em contato com o paradigma imperativo para depois serem abordados outros paradigmas e linguagens.

O paradigma imperativo é baseado na arquitetura de Von Neumann, que consiste, basicamente, de uma Unidade Central de Processamento (*Central Process Unit* - CPU) separada de uma memória. A memória armazena dados e instruções e a CPU executa uma única instrução por vez, de maneira sequencial, uma instrução seguida de outra. O projeto das linguagens imperativas é um reflexo dessa arquitetura, onde é possível encontrar estados, que representam células de memória em que dados são armazenados, ordens sequenciais e sentenças de atribuição, que consistem na alteração de dados presentes na memória. Os programas imperativos são sequências de instruções, ou ordens, para a realização de uma tarefa, e o resultado da execução desses programas é obtido através de mudanças de estados,

que, em outras palavras, seria através da manipulação de variáveis, que tratam-se de abstrações de células de memória. Em algumas linguagens, sequências de comandos podem ser nomeadas, identificadas por nomes, e esses nomes podem ser utilizados para invocar a sequência de comandos relacionada a tal nome. Essas sequências de comandos são chamadas de procedimentos, funções ou subprogramas. Linguagens que possuem essa característica são conhecidas como procedurais ou procedimentais.

A principal justificativa para se adotar a abordagem imperativo-primeiro é que o armazenamento de informações e o fato de se fazer uma coisa após outra, ou seja, a realização de atividades em passos sequenciais, fazem parte da vida cotidiana das pessoas e, portanto, os educadores acreditam que tais conceitos sejam de fácil entendimento. Porém, na prática, não é isso que é observado. Os alunos apresentam dificuldade no entendimento desses conceitos logo nos primeiros dias de aula. Outro problema encontrado é a dificuldade de posterior aprendizado de outros paradigmas de programação, sendo que o contato com outros paradigmas é de extrema importância para a formação profissional do aluno, principalmente a programação orientada a objetos, que se faz presente nas principais linguagens da atualidade e nas mais populares da indústria (BERGIN, 2000 apud VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008; DECLUE, 1996 apud VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008; GUZDIAL, 1995 apud VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008; LATTANZI; HENRY, 1996 apud VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008).

A dificuldade de se aprender outro paradigma e a importância e popularidade da programação orientada a objetos são as principais justificativas para a adoção da abordagem orientado-a-objetos-primeiro. Entretanto, aprender primeiro a programação orientada a objetos é difícil para os alunos, uma vez que exige habilidades de analisar e projetar antes de codificar. Além disso, autores temem que tal prática leve alunos a se tornarem designers, projetistas, sem a habilidade de codificação. A linguagem C++ é apontada como uma alternativa, pois permite a utilização de expressões de baixo nível, bem como expressões de alto nível. Porém, ao mesmo tempo, é complexa para a introdução à programação.

De outro lado, tem-se a abordagem conceitos-primeiro. Vujošević-Janičić e Tošić (2008) afirmam que tal abordagem "prevê uma precisa e concisa base para programar em todos os paradigmas" (p. 68). A exemplo disso, tem-se a abordagem da linguagem Kernel (VAN ROY; HARIDI, 2003), um subconjunto da linguagem multiparadigma Oz (MÜLLER; MÜLLER; VAN ROY, 1995). Tal prática é focada em conceitos subjacentes e, a partir desses conceitos, linguagens práticas que lidam com estes conceitos são selecionadas e traduzidas para o que os autores chamam de linguagens Kernel (ou linguagens núcleo), linguagens de

mais fácil entendimento, que “apresentam conceitos essenciais em uma maneira intuitiva e precisa.” (VAN ROY; HARIDI, 2003, p. 8). As linguagens Kernel apresentam uma semântica formal simples, ideal para programadores principiantes. A abordagem dá aos alunos uma visão ampla dos principais paradigmas de programação a partir de um framework uniforme. Vujošević-Janičić e Tošić (2008) ressaltam:

> Os paradigmas aparecem naturalmente, dependendo de quais conceitos básicos de programação são usados para o problema que deve ser resolvido. Assim, os alunos são capazes de situar os paradigmas em uma estrutura mais geral mostrando seus relacionamentos e como usá-los juntos. Parece que essa prática é muito acessível, porém não é muito difundida. (VUJOŠEVIĆ-JANIČIĆ; TOŠIĆ, 2008, p. 68).

Van Roy e Haridi (2003) destacam que os métodos atuais de ensino de programação, que partem da abordagem de um único paradigma, têm um efeito negativo na competência do programador e na qualidade dos programas. Em contrapartida, Müller, Müller e Van Roy (1995) afirmam que as linguagens multiparadigma são consideradas vantajosas para o ensino de algoritmos, uma vez que acrescentam pouca ou nenhuma complexidade quando algoritmos de diferentes classes são considerados.

Um outro exemplo do uso da programação multiparadigma para fins educacionais é a linguagem Leda (MÜLLER; MÜLLER; VAN ROY, 1995; VAN ROY; HARIDI, 2003). Budd (1995) destaca que o objetivo da criação dessa linguagem partiu da necessidade da criação de uma linguagem de programação “em que cada um dos diferentes principais paradigmas podem ser facilmente e naturalmente expressados, e em que cada paradigma se beneficie da presença dos demais.” (BUDD, 1995, p. 6).

CAPÍTULO 2
A ROBÓTICA SITUADA, A POO DE ALAN KAY, A FILOSOFIA LOGO E A PROGRAMAÇÃO MULTIPARADIGMÁTICA: "novos" olhares para o ensino de programação na Educação Básica

O presente capítulo disserta sobre as ideias centrais que permeiam e foram consideradas para a proposta de ensino de programação que se apresenta neste trabalho. São elas: robótica situada, orientação a objetos (OO) e as contribuições de Alan Kay, a filosofia LOGO e a programação multiparadigmática.

2.1 Robótica situada

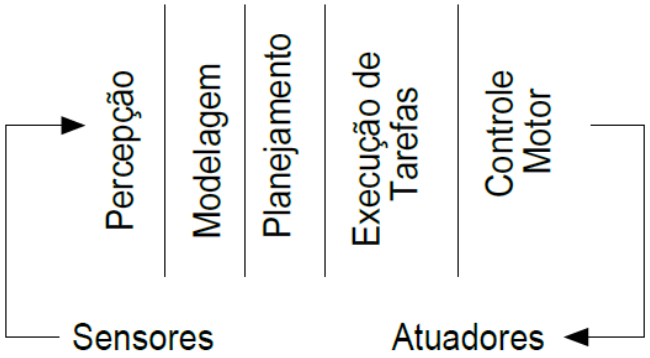
Em linhas gerais, a robótica situada se difere da robótica clássica no sentido em que a robótica clássica considera que o robô possui um mapa do mundo externo em sua mente, enquanto que, para a robótica situada, tal mapa é construído a partir da interação do robô, por meio de seu corpo, com o ambiente externo o qual está inserido (SOARES; BORGES; SANTOS, 2007). Tem-se aqui, dois conceitos fundamentais para a robótica situada, que são eles: o conceito da corporificação – o robô, um ser com corpo – e conceito o da incorporação – o robô, um ser incorporado a um ambiente. As teses da corporificação (*embodiment thesis*[15]) e da incorporação (*embedding thesis*) são duas das três ideias centrais da Cognição Situada. A Cognição Situada, por sua vez, fornece a base para a Robótica Situada (ROBBINS; AYDEDE, 2008). Tais ideias serão discutidas com mais detalhes na subseção 2.1.1.

Em meados da década de 80, iniciou-se um movimento rumo a uma mudança paradigmática das pesquisas em robótica e em inteligência artificial (IA). Esse movimento teve início com os trabalhos de Rodney Brooks, do *MIT AI Lab* (Laboratório de Inteligência Artificial do *Massachusetts Institute of Technology*). Brooks pregava que uma mudança paradigmática era necessária para que a IA prosperasse rumo a uma inteligência de máquina comparável à inteligência dos seres humanos. Na época, as pesquisas eram desenvolvidas

[15] Optou-se por acrescentar ao texto os termos originais, em inglês, "*embodiment thesis"* e "*embedding thesis"* para deixar claro ao leitor que no presente trabalho utiliza-se os termos "corporificação*"* e "incorporação*"* para se referir à "*embodiment"* e "*embedding"*, respectivamente, uma vez que que não foi identificado na literatura brasileira utilizada um consenso quanto às traduções usadas para ambos os termos. Alguns autores, por exemplo, utilizam "incorporação" para se referir a "*embodiment*", enquanto outros fazem uso do mesmo termo para se referir à "*embedding*". O mesmo ocorre para "corporificado" (com corpo) e "incorporado" (*embodied* e *embedded*, respectivamente). A autora achou pertinente utilizar termos encontrados na literatura que expressem melhor o significado de cada uma das teses e, também, a utilização de termos distintos para cada um dos termos originais, uma vez que tratam-se de ideias distintas entre si. No contexto em que são utilizados, corporificado se refere ao fato de que um ser humano, ou um robô, é dotado de um corpo, e incorporado, ao fato de que este humano/robô está inserido, imerso, em um mundo, um ambiente.

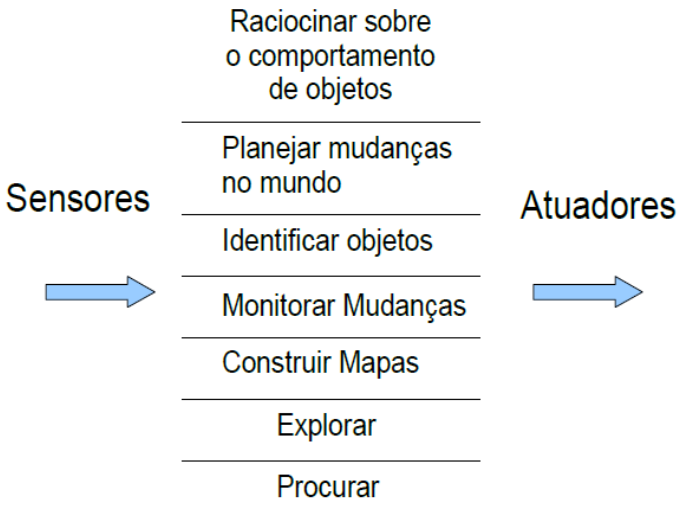
com base em problemas desvinculados do mundo real. As pesquisas em robótica eram realizadas considerando o "paradigma serial de processamento", ilustrado na Figura 1. Brooks propôs a substituição de tal paradigma por um paralelo, que foi chamado de "Arquitetura de Subsunção", retratada na Figura 2. A Arquitetura de Subsunção deu início, posteriormente, à "Robótica Baseada em Comportamentos".

Figura 1 – Paradigma serial de processamento



Fonte: GUDWIN, 2005, p. 3.

Figura 2 – Arquitetura de Subsunção



Fonte: GUDWIN, 2005, p. 3.

A Robótica Baseada em Comportamentos trouxe duas mudanças significativas: i) passou-se a descartar a ideia de que, para que um robô pudesse agir de forma inteligente, era necessário ter uma representação centralizada do mundo; ii) emerge a ideia de que o robô encontra-se "incorporado e em contato com o mundo" (GUDWIN, 2005, p. 3). A inteligência do robô passa a não ser vista mais como um processamento computacional, em que recebe-se uma entrada e gera-se uma saída, mas, ao invés disso, deve-se haver um componente

dinâmico que permita que o robô aja a partir de mudanças do ambiente em contato com seu corpo.

Gudwin (2005) destaca que, na mesma época, ocorria uma mudança também na ciência cognitiva, que, até então, se preocupava em explicar as funções cognitivas dos seres humanos. O cognitivismo passa a se basear na Autopoiese de Maturana e Varela (1991 apud GUDWIN, 2005): a cognição ocorre por meio da interação do indivíduo com o ambiente ao qual está inserido. Tal interação com o ambiente, por sua vez, ocorre por meio do corpo do indivíduo. Nas palavras de Gudwin, "uma mente só pode funcionar enquanto mente, caso tenha um corpo por meio do qual possa interagir com algum ambiente." (GUDWIN, 2005, p. 3).

Em um primeiro momento, cientistas optaram por tentar equiparar a inteligência de robôs à inteligência de insetos. Surge, então, os robôs insetos. A partir daí, começou uma caminhada rumo à construção de robôs humanoides. Neste contexto, pesquisadores do MIT apontaram 4 pontos fundamentais da inteligência humana a serem considerados em robôs humanoides: "o desenvolvimento incremental da inteligência, a interação do robô com seu ambiente, a interação social do robô e a interação multi-modal." (GUDWIN, 2005, p. 4). Considerando esses pontos, os robôs devem ser capazes de adquirir habilidades por meio da aprendizagem e da sua interação com o ambiente, da interação social e da interação multimodal. Para adquirir habilidades, a interação com o mundo por si só não é suficiente. A interação social também se faz importante. Aprendemos não somente com o ambiente, mas também com outros seres humanos. Já a interação multimodal está relacionada aos 5 sentidos – visão, audição, tato, paladar e olfato. Para Gudwin (2005), "nossos modelos internos dos objetos do mundo (...) são o resultado da interação de nossos sentidos." (GUDWIN, 2005, p. 5). Em outras palavras, nós interagimos com o ambiente por meio dos nossos sentidos.

Tem-se também aqui as ideias da mente incorporada e da mente corporificada, que, como dito anteriormente, tratam-se de duas das ideias centrais da Cognição Situada e que serão exploradas mais à frente. Gudwin (2005) disserta também sobre outros aspectos relacionados à Robótica Situada que se remetem à ideia da mente incorporada:

> "as representações internas podem ser lastreadas em interações sensório-motoras com o ambiente. Assim, o mundo pode ser visto como uma extensão da memória do robô, e este pode se servir dele para armazenar marcas e sinais que podem ser importantes em seu raciocínio." (GUDWIN, 2005, p. 4).

Na visão da robótica situada, portanto, um robô é uma criatura corporificada e incorporada. Ou seja, nas palavras de Haselager (2007), “os robôs possuem um corpo (...) por meio do qual interagem com o ambiente, constituído por objetos e outras criaturas (...) e sua imersividade no mundo decorre de seu comportamento e de seus processos cognitivos.” (HASELAGER, 2007, p. 253). Ademais, os comportamentos de um robô não são pré-determinados, mas sim emergentes. Emergente no sentido que o comportamento do robô não é programado diretamente, “mas surge das interações entre um número limitado de componentes que podem ser substancialmente diferentes em suas propriedades e possibilidades de ação.” (HASELAGER, 2007, p. 253). Neste sentido, portanto, é que surge a necessidade de um componente dinâmico, como destacado por Gudwin (2005).

Além disso, um dos pontos centrais da Robótica Situada, para o presente trabalho, é a interação do robô com objetos do mundo real. Em termos de programação, isso nos remete a um dos paradigmas mais populares: a programação orientada a objetos, que será tratada da seção 2.2.

2.1.1 Cognição Situada

As abordagens para explicar a cognição podem ser divididas em duas vertentes. A primeira entende que “a realidade é pré-dada e independente do sujeito” (VENÂNCIO; BORGES, 2006, p. 30). Tal abordagem é tida como objetiva. Sendo assim, as abordagens objetivistas separam sujeito e objeto em “mundo das coisas” e “mundo da mente”, e considera que o mundo é objetivo e que suas características são captadas e representadas na mente do ser humano. O indivíduo é apenas um observador neste processo, cabendo a ele a tarefa de somente recuperar as características adequadamente. Já a segunda vertente, entende que a realidade é “construída pelo sujeito no seu curso de interação com o ambiente” (VENÂNCIO; BORGES, 2006, p. 30) e é, portanto, considerada não-objetiva. Dentre as abordagens não-objetivas, está a Cognição Situada. Tal abordagem não separa sujeito e objeto, pois possui a visão de que a construção do objeto é algo que depende do sujeito. Cada ser humano constrói o seu próprio mundo, a medida em que interage com o que está ao seu redor. Venâncio e Borges (2006) salientam ainda que “organismo e ambiente constituem uma unidade inseparável, e a dinâmica de interação ocorre contínua e simultaneamente.” (VENÂNCIO; BORGES, 2006, p. 32).

Conforme apresentado brevemente na subseção anterior, três teses são centrais para a Cognição Situada: a tese da corporificação (*the embodiment thesis*); a tese da incorporação

(*the embedding thesis*); e a tese de extensão (*the extension thesis*). Robbins e Aydede (2008) trazem uma breve descrição para cada uma delas: tese da corporificação – "cognição depende não somente do cérebro, mas também do corpo" (ROBBINS; AYDEDE, 2008, p. 3; tese da incorporação – "atividade cognitiva rotineiramente explora estrutura no ambiente natural e social" (ROBBINS; AYDEDE, 2008, p. 3; tese de extensão – "os limites da cognição se estendem além dos limites de organismos individuais" (ROBBINS; AYDEDE, 2008, p. 3). As subseções a seguir dissertam brevemente sobre cada uma dessas três teses.

2.1.1.1 A mente incorporada

A ideia da mente incorporada se opõe à visão da cognição para a Ciência Cognitiva clássica. Para a Cognição Situada, a cognição não ocorre meramente por meio de processamentos internos de informações, a partir de determinadas entradas. Ao contrário disso, o cérebro explora o ambiente externo, a fim de otimizar seu armazenamento interno. O cérebro descarrega constantemente seu trabalho cognitivo no ambiente. Ao invés de manter armazenadas representações complexas do mundo exterior, ele, de maneira constante, procura no ambiente externo as informações necessárias no momento em que as precisa. Portanto, para a Cognição Situada, é fundamental o fato de que a mente corporificada está inserida em um ambiente, uma vez que o processo cognitivo depende diretamente do ambiente em que o sujeito está inserido. A cognição não ocorre unicamente por meio de processamentos internos, mas também por meio da interação do sujeito com seu ambiente externo.

Outro ponto importante para a teoria da mente incorporada é que o ser humano não aprende somente em contato com objetos, mas também, como dito anteriormente, em contato com outros seres humanos. A interação social é também um fator importante para o processo cognitivo.

2.1.1.2 A mente corporificada

Robbins e Aydede (2008) destacam a importância do corpo para a cognição. Entender o significado do símbolo "cadeira", por exemplo, requer não somente ser capaz de reconhecer o objeto cadeira, mas também compreender o significado de sentar, identificar quais movimentos são necessários para se sentar e distinguir em quais objetos se pode sentar ou não (ANDERSON, 2003 apud ROBBINS; AYDEDE, 2008). Nossas capacidades motoras e sensoriais não dependem somente do trabalho do cérebro e da medula espinhal, depende

também dos “órgãos sensoriais, do sistema musculoesquelético e partes relevantes do sistema nervoso periférico (como nervos sensoriais e motores).” (ROBBINS; AYDEDE, 2008, p. 4).

É importante destacar também que a cognição pode ser corporificada de duas maneiras: online e off-line. A cognição em uma visão corporificada online se dá pelas interações que ocorrem entre o cérebro sensório-motor com partes externas do corpo. Já em relação à corporificação off-line, trata-se da “dependência da função cognitiva com áreas sensório-motoras do cérebro mesmo na ausência de entrada sensorial e saída motora.”, ou seja, dos processamentos internos realizados pelo cérebro (ROBBINS; AYDEDE, 2008, p. 4).

A visão da mente corporificada, portanto, está relacionada à dependência da mente com o corpo. A mente só é mente inserida em um corpo. A cognição é dependente da interação do indivíduo com o ambiente externo e tal interação ocorre por meio do corpo, por meio de processos sensório-motores.

2.1.1.3 A mente estendida

Dentre as três teorias que embasam a Cognição Situada, a teoria da mente estendida é a que mais se opõe à visão da Cognição Clássica. Na visão da mente estendida, a atividade cognitiva é dividida entre o indivíduo e o mundo externo. A mente é, portanto, expandida, ela ultrapassa seus próprios limites. Nessa visão, a mente não reside apenas no universo delimitado pelo cérebro. Ao invés disso, ela ultrapassa esses limites, ela expande para o ambiente externo.

A teoria da mente estendida não é amplamente aceita, porém se torna natural pensar que a mente se estende para além do corpo ao considerar que o sistema cognitivo se altera à medida em que tanto ambiente interno, quanto externo, sofrem mudanças (ROBBINS; AYDEDE, 2008).

2.1.2 A Robótica Situada e a programação

As ideias centrais da robótica situada podem ser incorporadas à programação. Em uma visão tradicional, um programa recebe entradas, as processa e gera saídas. Em uma visão situada, deve-se considerar que um programa está inserido em um ambiente ao qual a programação ocorre. O ambiente ao qual o programa está inserido, ou seu estado atual, influenciam diretamente nas ações a serem executadas pelo programa. Entende-se que tal

analogia ficará mais clara para o leitor no Capítulo 3, em que a proposta do presente trabalho é, de fato, apresentada.

2.2 Programação Orientada à Objetos

Alan Kay é considerado um dos pais da programação orientada a objetos (POO). Em 1970, Kay ingressou para o Centro de Pesquisa de Palo Alto (PARC – *Palo Alto Researsh Center*), da Xerox, e, então, trabalhou na criação da linguagem Smalltalk, considerada a primeira linguagem puramente orientada a objetos. Smalltalk foi projetada para o sistema Dynabook, um computador portátil, idealizado por Kay, voltado para o uso por crianças. (KAY, 1996). É importante destacar que o Dynabook antecede a criação dos computadores pessoais e é pioneiro dos dispositivos móveis, como o notebook. E, no que concerne o presente trabalho, cabe salientar ainda que o trabalho de Seymour Papert com crianças e a programação LOGO teve forte influência nos trabalhos de Kay. Desde seu contato com Papert e LOGO, o autor procurou desenvolver um computador voltado para a aprendizagem das crianças. Para ele, o uso do computador poderia mudar a maneira com que as crianças aprendiam e essa aprendizagem proporcionada pela máquina se dava pela simulação. Entender o Teorema de Pitágoras, por exemplo, através da apresentação de sua fórmula é mais complexo que através de uma simulação/animação. Para Alan Kay, um computador vai além de um livro. Com um computador, o usuário pode ser leitor e autor ao mesmo tempo (MORAIS, 2016).

A programação orientada a objetos é um paradigma de programação que permite ao programador se concentrar no sistema e em suas entidades, que tratam-se de objetos do mundo real representados no sistema, e em como essas entidades devem se relacionar entre si e não nos procedimentos a serem realizados para se atingir determinado objetivo. As principais características da programação orientada a objetos são composição de classes e objetos, instanciação de objetos, polimorfismo e encapsulamento. (KAY, 1996).

Tucker e Noonan (2008) afirmam que a POO surgiu da necessidade de melhoria dos tipos abstratos de dados (TAD). Um exemplo de um TAD seria um tipo Fila. Uma fila, em termos computacionais, é uma estrutura de dados que permite agrupar dados em disposição de uma fila. Desse modo, ao inserir um dado em uma fila, esse dado passa a ocupar a última posição da fila e, ao remover um dado da fila, será removido o elemento que ocupa a primeira posição da fila (o primeiro que entra é o primeiro que sai – *First In, First Out* (FIFO)). Um tipo Fila encapsula a representação de uma fila e as operações necessárias a essa fila, como

remover e inserir. Uma classe pode ser vista como uma extensão ao conceito de um tipo abstrato de dados. Tucker e Noonan (2008) definem classe como "uma declaração de tipo que encapsula constantes, variáveis e funções para manipulações dessas variáveis." (TUCKER; NOONA, 2008, p. 315). A introdução da ideia de classe foi inserida para resolver problemas antes tidos com os tipos abstratos de dados.

Como veremos mais adiante, a ideia da programação orientada a objetos estava presente nos projetos da linguagem Simula, nos anos 60, e da Smalltalk, na década de 70, porém a POO só começou a ganhar popularidade na década de 80 com aplicações práticas, voltadas principalmente para o desenvolvimento de Interfaces Gráficas de Usuários (*Graphic User Interfaces* – GUIs). Uma aplicação GUI é mais naturalmente projetada quando os componentes da interface gráfica, como botões e campos de texto, são vistos como objetos que interagem entre si e trocam informações. A programação orientada a objetos passa, então, a ter um foco na composição de objetos e não em procedimentos e abstrações de dados (SEBESTA, 2011; TUCKER; NOONA, 2008).

Em "*Early history of Smalltalk*", Kay (1996) descreve como a programação orientada a objetos veio em sua mente, assim como de que maneira se deu a linguagem Smalltalk e o Dynabook, o que influenciou seus projetos e os caminhos que percorreu para que chegasse a tais soluções, bem como um breve relato de seu trabalho com crianças. Além disso, aponta características da POO que considera importante.

2.2.1 Uma breve história de Smalltalk e as ideias centrais da Programação Orientada a Objetos

Inicialmente, a ideia da Programação Orientada a Objetos (POO) veio à mente de Alan Kay ainda no início da década de 60, quando era programador da Força Aérea. Kay aponta duas motivações centrais para a POO: "encontrar um esquema modular melhor para sistemas complexos envolvendo ocultamento de detalhes" e "encontrar uma versão mais flexível de atribuição e depois eliminá-la por completo" (KAY, 1996, p. 514). Entretanto, a ideia somente tomou forma mais clara para ele anos depois quando teve contato com a máquina Sketchpad e a linguagem Simula, em 1966. Isso se deu quando Alan Kay, enquanto estudante de pós-graduação da Universidade de Utah, foi ao escritório de Dave Evans a procura de emprego. Evans entregou-lhe um texto de título "Sketchpad: Um sistema gráfico de comunicação homem-máquina". Sketchpad tratava-se do primeiro sistema a ter janelas e em que o usuário poderia interagir graficamente. "Foi a invenção da computação gráfica moderna

e interativa", salienta o autor (KAY, 1996, p. 515). Além disso, em Sketchpad, havia um "desenho mestre", que, a partir dele, era possível gerar "instâncias de desenhos" (KAY, 1996, p. 515).

Naquele momento, Alan Kay começa a trabalhar com Dave Evans e sua primeira tarefa foi fazer a (linguagem) "Algol para 1108" (KAY, 1996, p. 516) funcionar – 1108 tratava-se de um computador. Naquela época, não haviam ainda os sistemas operacionais e a interação com a máquina era feita através de uma linguagem de programação. Linguagens de programação eram projetadas, portanto, para máquinas específicas e cada máquina rodava uma única linguagem. Depois de se "debruçar" sobre o projeto da linguagem, juntamente com outro estudante de pós-graduação, chegaram à conclusão de que se tratava da Algol para 1107, modificada para se criar uma linguagem chamada Simula. O que mais chamou à atenção de Kay foi que as estruturas de Simula eram muito parecidas com as de Sketchpad. Em Simula, havia a presença de atividades e processos, que, em Sketchpad, tratavam-se dos mestres e instâncias. As ideias que tivera em 61, se faziam mais claras e concretas em sua mente naquele momento. Para expressar tais ideias, o autor cita uma frase de Bob Barton (apud KAY, 1996, p. 516): "O princípio básico do design recursivo é fazer com que as partes tenham o mesmo poder que o todo." Kay considerou o computador inteiro como sendo o todo e se questionou por que, ao invés de dividi-lo em partes menores, mais fracas, como em estrutura de dados e procedimentos, por que não o dividir em o que chamou de "pequenos computadores" (KAY, 1996, p. 516), com o mesmo potencial do todo?

Simula foi, portanto, a grande influência de Smalltalk, que seria criada somente anos depois. Entre estes dois acontecimentos, Alan Kay veio a trabalhar ainda, entre os anos de 1967 e 1969, com Ed Cheadle, conhecido como um "gênio dos hardwares amigáveis" (KAY, 1996, p. 517). Kay e Cheadle trabalharam juntos na construção de uma pequena máquina chamada FLEX. A máquina não se tratava do primeiro computador pessoal, mas Cheadle gostaria que fosse um computador para "profissionais não-computacionais" (KAY, 1996, p. 517), ou seja, que fosse de uso de pessoas sem conhecimentos técnicos em Computação – como, na época, a interação com a máquina se dava por meio de uma linguagem de programação, somente quem tinha conhecimentos em computação conseguia usá-las. Dessa forma, seu objetivo era que fosse utilizada para a máquina uma linguagem de alto nível. O autor, então, sugeriu o uso da linguagem JOSS. JOSS, porém, não atendia à tudo aquilo que queriam e Simula era "grande" demais para a pequena máquina. O ponto positivo que Kay salienta de JOSS, e que, inclusive, serve de inspiração a ele posteriormente, era a sua "extrema importância com seu design para o usuário final" (KAY, 1996, p. 517), porém era

muito lenta. Nesse sentido, trabalharam no projeto de uma linguagem que ficaria conhecida como FLEX.

Ainda em 1967, Alan Kay teve contato com os trabalhos de Doug Engelbart, considerado um dos pais da computação pessoal. Engelbart idealizava que os sistemas lineares (NLS), no futuro, seriam o "'aumento do intelecto humano' através de um veículo interativo navegando por meio de 'vetores de pensamento no espaço conceitual'" (KAY, 1996, p. 518). Kay utilizou muitas das ideias de Engelbart no projeto da máquina FLEX e já previa, na época, a popularização dos computadores pessoais: "ao invés de 4000 mainframes IBM em todo o mundo e, no máximo, milhares de usuários treinados para cada aplicativo, haveriam milhões de máquinas e usuários, principalmente fora do controle institucional direto" (KAY, 1996, p. 518).

No ano de 67 também, em uma das reuniões realizadas pela ARPA para os alunos de pós-graduação, reunião essa cujo assunto era educação, Kay teve seu primeiro contato com Marvin Minsky. Minsky discursava de forma contrária aos "métodos tradicionais de ensino" (KAY, 1996, p. 522) e foi quando teve também seu primeiro contato com as ideias de Piaget e Papert.

No ano seguinte, em 1968, os estudantes de pós-graduação se reuniram em Illinois e Kay palestrou sobre a máquina FLEX. Durante a conferência, o que mais lhe chamou a atenção ocorreu durante uma visita à Universidade de Illinois, onde viu pela primeira vez um monitor de tela plana. Ele passou o resto da conferência pensando como isso poderia funcionar na máquina FLEX.

Ainda em 68, ao conhecer GRAIL, Kay concluiu que a interface de FLEX não estava de acordo com o que desejavam. Neste ano também, ele foi ao encontro de Seymour Papert e os outros pesquisadores que construíram LOGO – "Aqui estavam crianças fazendo programação real com uma linguagem e ambiente especialmente projetados", ressalta o autor (KAY, 1996, p. 523). Da mesma forma em que Simula lhe inspirou a idealizar a POO, naquele momento Kay teve em sua mente uma imagem de como a computação pessoal seria, ou deveria ser. Para ele, a computação pessoal não se tratava de um "veículo pessoal dinâmico", mas de um "meio pessoal dinâmico". "Com um veículo, era possível esperar até o ensino médio e dar "motoristas", mas, se fosse um meio, ele teria que se estender ao mundo da infância." (KAY, 1996, p. 523).

Juntas, todas aquelas experiências com a máquina FLEX, GRAIL, o monitor de tela plana, o discurso de Barton e o trabalho de Papert com crianças levaram à ideia de como o computador pessoal deveria ser de fato para Alan Kay. Ele salienta que deveria ser pequeno,

com uma interface amigável, como de JOSS, GRAIL e LOGO, porém com o poder de Simula e de FLEX. Kay trabalhou, então, no aprimoramento da máquina e linguagem FLEX até a defesa de sua tese, em 1969. Uma das evoluções conceituais da POO alcançadas por ele nessa época foi que, em sua mente, os objetos, vistos como minicomputadores, deveriam interagir entre si através da troca de mensagens. O que cada objeto fazia internamente não deveria ser de interesse e conhecimento dos outros objetos.

Após a defesa de sua tese, Kay foi para o projeto de Inteligência Artificial (IA) de Stanford, onde passou grande parte do tempo trabalhando com o notebook KiddyKomputers, em vez de trabalhar de fato com IA. E, no final do ano de 1969, Kay observou uma característica de LISP que lhe chamou à atenção: em LISP, uma linguagem considerada puramente funcional, seus principais recursos, como expressões lambda, não se tratavam de funções, mas, ao contrário disso, eram tratados como "formas especiais" (KAY, 1996, p. 524). Isso o levou a uma ideia que seria posteriormente aplicada à Smalltalk: "pegue a coisa mais profunda e difícil que você precisa fazer, torne-a ótima e então construa tudo com mais facilidade." (KAY, 1996, p. 524). Para Kay, objetos eram a "coisa mais profunda e difícil" (KAY, 1996, p. 524) da POO.

Em meados do ano de 1970, Alan Kay começa a trabalhar no PARC, em uma versão de KiddiKomp, que seria, posteriormente, descrito por ele como Dynabook. O autor relata alguns conflitos de pensamento entre Xerox e um grupo de pesquisadores que haviam vindo da ARPA, o que o motivou, no ano seguinte, orientado por Bill English, a iniciar um grupo, que chamou de Grupo de Pesquisa de Aprendizagem (*Learning Research Group* – LRG). Ele relata ainda ter contratado para o grupo "somente pessoas que tiveram estrelas nos olhos ao ouvir sobre a ideia do notebook" (KAY, 1996, p. 527). Os integrantes do grupo se tornaram muito próximos uns com os outros e discutiam constantemente os objetivos e potenciais do Dynabook. Tinham como principal objetivo levar à população uma nova forma de pensar.

Em 71, também, a ideia do KiddiKomp foi melhorada e tinha, dentre outras coisas, uma linguagem que Kay chamou de Smalltalk (conversa fiada, em português) – "como em 'programação deve ser uma questão de ...' e 'crianças devem programar em ...'" (KAY, 1996, p. 528). O autor cita que Smalltalk foi influenciada pelas linguagens FLEX, PLANNER, LOGO, META II e as próprias derivações delas criadas por ele. Kay estava interessado em que as coisas fossem claras o bastante para que "crianças pudessem entender após alguns poucos anos de experiência com programação mais simples" (KAY, 1996, p. 528). Deveria ser um sistema "simples e pequeno", porém que permitisse, ao mesmo tempo, fazer "coisas interessantes" (KAY, 1996, p. 530). Nesse contexto, Kay destaca que linguagens de máquinas

universais podem ser projetadas utilizando-se algumas poucas expressões genéricas ao ponto que possam definir “qualquer coisa que possa ser calculada”, porém muitos não as considerariam “bonitas” (KAY, 1996, p. 530).

Nessa época, Kay evoluiu também as ideias por trás da “linguagem para as crianças” (KAY, 1996, p. 531), principalmente embasado por Piaget e Jerome Bruner. Ao ler tais autores, se preocupou com a abordagem mais simbólica utilizada por FLEX, LOGO e a própria Smalltalk, pois as crianças ainda estavam começando a entrar no estágio simbólico. Ao fazer um planejamento para os anos seguintes, Kay colocou então, nesse planejamento, o que chamou de “programação icônica”. O autor utiliza esse termo, pois, de acordo com os estágios do desenvolvimento de uma criança, apontados por Bruner, o estágio anterior ao estágio simbólico é chamado por Bruner de estágio icônico, e ocorre entre os 4 e 8 anos (KAY, 1972). De acordo com Kay (1972), o estágio icônico seria correspondente ao estágio concreto-operacional, apontado por Piaget.

Em 1972, foi lançada a primeira versão de Smalltalk e do Dynabook. Àquela época, Smalltalk era o único sistema executado pelo Dynabook. Um dos diferenciais do Dynabook era sua interface gráfica, que, como evidenciado por ele, era uma das prioridades do projeto. O interessante é que o Dynabook apresentava características revolucionárias para a época, todas voltadas para crianças, como arquivos multimídia, música, ferramenta para desenho, entre outros. E, em Smalltalk, até mesmo as classes eram instâncias de si mesmas. Além disso, Kay acreditava que seria mais fácil convencer os professores e conselhos escolares sobre o Dynabook, apresentando-o como um substituto ao livro do que apresentar-lhes os reais ganhos com a visão que tinha da computação pessoal.

Um dos recursos adicionados posteriormente ao Dynabook foi uma “versão orientada a objetos da tartaruga LOGO” (KAY, 1996, p. 538). Com essa versão de LOGO, era possível que várias tartarugas fossem criadas e não somente uma. Existia uma classe de “tartaruga comandante que podia controlar uma tropa de tartarugas” (KAY, 1996, p. 538). Kay cita ainda o projeto PYGMALION, desenvolvido nessa época por Dave Smith, que tratava-se de um “ensaio da programação icônica” (KAY, 1996, p. 540). A partir daí, Smalltalk evolui para versões posteriores, como Smalltalk-76.

É evidente que o que Kay idealizou por programação orientada a objetos está longe de ser o que se tem visto nas linguagens que se baseiam em tal paradigma. Um ponto, inclusive destacado por ele, é que, como citado anteriormente, a segunda das duas motivações para POO seria eliminar as operações de atribuição, porém as linguagens orientadas a objetos fazem uso excessivo de operações de atribuição. Nesse sentido, tais linguagens não são

consideradas puramente orientadas a objetos. A maioria dessas linguagens combina as características da POO – composição de classes, instanciação de objetos, encapsulamento, etc. – com características da programação imperativa, como operações de atribuição e variáveis. A própria definição de classe apresentada por Tucker e Noonan (2008) traz consigo o termo "variáveis". Kay acreditava que "os programadores humanos não são máquinas de Turing e quanto menos seus sistemas de programação requererem técnicas de máquina de Turing, melhor." (KAY, 1996, p. 543). Para se compreender o significado de uma operação de atribuição, por exemplo, é necessário compreender a arquitetura de hardware de um computador. Nota-se, portanto, nesse caso, uma dependência conceitual entre máquina e programação. Operações de atribuição também não estão presentes em linguagens puramente funcionais. A expressão $x = x + 1$, por exemplo, tem significados completamente distintos para a programação imperativa e para a programação funcional. Além disso, o autor aponta uma diferenciação conceitual de objeto e uma noção fraca de encapsulamento apresentada pelas linguagens.

2.2.2 Smalltalk e o ensino de programação a crianças

Kay e sua equipe iniciaram os trabalhos com crianças em 1973. Eles nunca haviam trabalhado com crianças antes e, por isso, convidaram Adele Goldberg e Steve Weyer para auxilia-los nesse projeto. Em uma primeira atividade, Adele utilizou algo semelhante aos "gráficos de tartaruga LOGO" (KAY, 1996, p. 543). Como, naquela época, a computação pessoal lidava com ferramentas interativas, Kay acreditava que a "alfabetização" (KAY, 1996, p. 544) deveria se dar através da "criação de ferramentas interativas pelas crianças" (KAY, 1996, p. 544).

Posteriormente, Adele utilizou o "Joe Book", sugerido por ela, para ensinar Smalltalk como uma linguagem orientada a objetos. O *Joe Book* consistia em ensinar uma linguagem de programação através de exemplos de programas sérios. Animações simples eram geradas criando-se instâncias da classe "caixa" e enviando mensagens a essas instâncias. Exemplo: ao usar o comando "box new named 'Joe'!" uma instância da classe caixa chamada "Joe" era criada. Ao enviar a mensagem "Joe erase", a caixa que havia sido instanciada anteriormente, de nome "Joe", era apagada, ou, ao enviar a mensagem "Joe show", a caixa "Joe" era exibida. Depois disso, passava-se para um etapa em que as crianças deveriam adivinhar como deveria ser determinada classe de caixa e, ao final, a tal classe de caixa era mostrada a elas. Muitos projetos interessantes surgiram das classes de caixas, porém não de maneira generalizada.

Em 74, Alan Kay ensinou Smalltalk para 20 pessoas da PARC que não sabiam programar. Embora eles tenham conseguido assimilar o material introdutório mais rapidamente que as crianças, ainda assim se depararam com problemas que não sabiam resolver. Kay conclui com isso que “não é suficiente apenas aprender a ler e escrever. Há também uma literatura que gera ideias.” (KAY, 1996, p. 545). E, então, Adele compreendeu que era necessário uma etapa intermediária entre “as ideias vagas sobre o problema” (KAY, 1996, p. 545) e a codificação. As crianças passaram, assim, a primeiramente analisar algo que gostariam de criar computacionalmente, um problema, e passavam para a decomposição do problema em classes e mensagens e, posteriormente, para a descrição, em linguagem humana, do planejamento que imaginavam que deveriam ser os métodos. Essa descrição depois seria um comentário e guiaria a codificação dos métodos. E a ideia funcionou, porém nada poderia se concluir fortemente disso, visto fora utilizada em um pequeno grupo de crianças. Uma outra abordagem utilizada foi deixar as crianças trabalharem a partir de estruturas mais complexas criadas por programadores mais experientes.

Para Kay, o problema não estava na linguagem a ser usada ou nos recursos que possuem a linguagem a ser usada para o ensino de programação, mas sim na construção ao longo de anos de “estruturas que precisam estar lá” (KAY, 1996, p. 546) para que pessoas sejam capazes de resolver determinados problemas. Isso remete às diversas situações em que alunos chegam à graduação e se deparam com determinados problemas/dificuldades de aprendizagem. Isso reforça que talvez tenha faltado a esses alunos a construção, ao longo dos anos anteriores, de tais estruturas. E reforça a necessidade de se investir e incentivar o ensino de programação na educação básica, mas não o ensino voltado à aprendizagem de linguagens e/ou técnicas específicas, mas sim ao ensino voltado à construção das “estruturas que precisam estar lá”.

Kay disserta que “o conhecimento está em seu estágio menos interessante quando é aprendido pela primeira vez” (KAY, 1996, p. 548). Nesse estágio, representações são barreiras que atrapalham a significação do aprendizado e devem ser desfeitas. É possível, posteriormente, percorrer dois caminhos, para ele, “importantes e interligados” (KAY, 1996, p. 548). O primeiro caminho se refere à fluência, que consiste em “construir estruturas mentais que fazem as interpretações das representações desaparecerem” (KAY, 1996, p. 548). Nesse processo, as palavras e caracteres de uma expressão são tratados como significações e não marcações, “a raquete de tênis ou o teclado se tornam uma extensão do próprio corpo, e assim por diante” (KAY, 1996, p. 548). Se prosseguir nesse caminho, o ser humano em questão se torna um especialista, mas sem conhecimento especializado em outras áreas e as

generalizações são possíveis, porém mal formuladas. O outro caminho é “tomar o conhecimento como uma metáfora do que pode iluminar outras áreas” (KAY, 1996, p. 548). Mas, ao percorrer esse caminho, sem fluência, é provável que as metáforas sejam “confusas e enganosas” (KAY, 1996, p. 548), sejam ofuscadas por conhecimentos mais consolidados. Nota-se, portanto, como os dois caminhos são interdependentes. É preciso, portanto, ser “fluente e profundo enquanto constrói relacionamentos com outros conhecimentos fluentes profundos” (KAY, 1996, p. 548). O autor acredita que alfabetização não se trata de “ser capaz de ler um aviso em uma caixa de remédios” ou de “escrever sobre as férias de verão” (KAY, 1996, p. 548), mas sim de ser capaz de ler o *Common Sense* de Paine e escrever uma crítica sobre isso. Neste contexto, a seguir, destaca-se um trecho em que Kay esclarece porque muitos, assim como ele, acreditam ser importante que crianças entendam computação:

> “A razão, portanto, que muitos de nós queremos que as crianças compreendam a computação profunda e fluentemente é que, como literatura, matemática, ciência, música e arte, ela traz maneiras especiais de pensar sobre situações que, em contraste com outros conhecimentos e outras formas de pensar criticamente, impulsiona nossa capacidade de entender nosso mundo.” (KAY, 1996, p. 548)

Por fim, o autor tinha uma visão da computação pessoal diferente da computação pessoal que lidamos hoje. A computação pessoal hoje é vista como a computação presente no cotidiano dos seres humanos, nas tecnologias digitais desenvolvidas pela Computação, as quais a população em geral faz uso. Contrário a isso, a visão que Alan Kay tinha da computação pessoal vai de encontro ao que Seymour Papert propunha para o uso dos computadores, que será discutido na próxima seção.

2.3 A filosofia LOGO

Seymour Papert, na década de 60, trabalhou na construção de um ambiente de aprendizagem para crianças, que chamou de LOGO. Tratava-se de as crianças aprenderem matemática e programação dando comandos a uma tartaruguinha, que, a partir de tais comandos, fazia “rabiscos”. O objetivo era as crianças criarem desenhos, como formas geométricas, por exemplo. Como mencionado no primeiro capítulo, na época, as ideias de Papert tiveram pouca repercussão, visto que o computador ainda não era um instrumento popular. Com a popularização dos computadores e, principalmente, com o aquecimento da discussão sobre o ensino de programação na educação básica, suas ideias vieram a se tornar

referência para muitos pesquisadores da área. Como visto na seção anterior, Papert foi, inclusive, naquela época, influência para os trabalhos de Alan Kay.

Influenciado por Piaget, Papert não tratava meramente do ensino de programação às crianças. Ele tratava, na realidade, de propiciar às crianças uma forma de aprendizagem diferente da habitual, que ocorria por meio da interação com o computador e, principalmente, em que a criança desempenhava um papel ativo nesse processo. Nesse sentido, suas ideias e trabalhos trazem inúmeras e riquíssimas contribuições para a educação e para as discussões em torno do ensino da computação para crianças e adolescentes. Suas ideias ficaram conhecidas como a filosofia LOGO.

Descrevendo um pouco o ambiente de programação LOGO, tem-se uma ferramenta computacional que possui uma janela gráfica e uma janela de comandos. Na janela gráfica, é possível visualizar a tartaruga, que trata-se do ator principal do ambiente de programação e é quem desempenhará a tarefa dada pela criança, e é possível ver também os resultados da execução dos comandos dados à tartaruga. A tartaruga possui um lápis e, ao dar à tartaruga o comando “parafrente 50”, por exemplo, a tartaruga andará 50 passos em linha reta, fazendo um risco em linha reta do local onde partiu até o local em que irá parar após os 50 passos. Para se desenhar um quadrado, será necessário, depois disso, ordenar que a tartaruga gire 90º à direita ou à esquerda e, posteriormente, ande mais 50 passos para frente, e assim por diante, até que a tartaruga volte ao local inicial, fechando, assim, o quadrado. É possível também ordenar que a tartaruga não use o lápis, de modo que ela passará a andar sem fazer um risco. E é possível fazer com que ela use uma borracha, apagando os riscos por onde passa ou fazer também com que se mude a cor do lápis, de modo a criar um desenho colorido (PAPERT, 1985). Cabe destacar que, com o avanço do computador, diversos autores criaram ferramentas baseadas no sistema LOGO descrito por Papert, de modo a obter uma ferramenta que rode no ambiente que se dispunha, além de permitir utilizar comandos na língua nativa do experimento, como o Português, por exemplo.

Ao se trabalhar o ensino da programação com a ferramenta LOGO, os alunos rapida e naturalmente percebem que, para se criar um desenho, é necessário organizar seus pensamentos estruturalmente, de modo que, se elabore uma sequência de comandos, que, ao ser executada, resultará na obtenção do desenho desejado. Eles compreendem sozinhos, explorando a ferramenta, que, para que a programação funcione, devem pensar em todos os passos sequenciais que devem ser realizados pela tartaruga. Esse é um conceito fundamental da programação, pelo menos da programação imperativa, e que é inicialmente ensinado em qualquer curso introdutório de programação.

O que Papert explora, portanto, é o aprender sozinho, explorando o ambiente. É o "aprender" e não o "ser ensinado". E essa é a principal crítica que o autor faz à escola. Para ele, a escola se preocupa muito com o ensino e pouco com o aprendizado. Considerando os estágios da aprendizagem apontados por Piaget, a criança em seus primeiros meses de vida, aprende, quase que unicamente, por meio do seu contato com o ambiente externo, de maneira que se dá pela exploração de tal ambiente, por meio da experiência. Nesse estágio, o contato com os pais/responsáveis tem pouca ou nenhuma influência no processo de aprendizagem da criança. É somente no próximo estágio que a criança começa a aprender por meio da troca de informações com os adultos. E, para Seymour Papert, esse estágio é o mais perigoso, pois a criança passa a ser ensinada a partir daquilo que lhe é contado. E esse estágio se torna mais expressivo quando a criança passa a ir para a escola. A criança passa a não ter mais a autonomia sobre o que lhe é ensinado e, muito menos, sobre como lhe é ensinado, e, na maioria das vezes, perde o interesse em aprender, em explorar. Perde, muitas vezes, sua criatividade, seu instinto curioso, explorador. Papert menciona que, os que sobrevivem ao segundo estágio, se tornam pessoas criativas em seus laboratórios e que essas pessoas são muito parecidas com as crianças que exploram.

O que ele propõe é contornar esse segundo estágio, não no sentido de pular tal estágio, mas no sentido de permitir que a criança passe por ele, mas sem perder sua capacidade de aprender, digamos. A proposta é de que o aluno seja ativo nesse processo e, para chegar à sua ideia de como o computador pode ser uma ferramenta poderosa nesse sentido, Seymour Papert, em uma conversa com Paulo Freire, em 1995, usa de exemplo algo que observou com seu neto. Um dia, ele observou seu neto colocar, sozinho, uma fita cassete em um aparelho de videocassete e operar o aparelho de modo a assistir ao vídeo da fita cassete, que era sobre máquinas de fazer estrada. Depois de um tempo, andando de carro com o neto, ao passar por uma daquelas máquinas, seu neto começou a fazer perguntas inteligentes que ele mesmo não sabia responder e nem mesmo teria a capacidade de formula-las, visto que o menino havia adquirido um conhecimento assistindo ao vídeo que ele mesmo não tinha. A diferença dele para seu neto é que, na idade de seu neto, ele apenas poderia adquirir aquele tipo de conhecimento perguntando aos adultos. As tecnologias digitais permitem, então, que as crianças e os adolescentes contornem o segundo estágio, o que, para ele, os poupa do trauma do ensino escolar, e permite manter a curiosidade experimental dessas crianças e adolescentes (PRIOLLI; RAMOS, 1995).

Uma das críticas que Papert faz é quanto ao uso do computador como um mero transmissor de conhecimento e, ao invés disso, propõe que a criança aprenda ao ensinar o

computador sobre como deve proceder para realizar determinada tarefa. É a “criança inteligente” ensinando o “computador burro”. Neste contexto, o autor desenvolve duas ideias: a de que programar é um processo natural de aprender a se comunicar com o computador e de que aprender a se comunicar com o computador pode modificar a maneira como outras aprendizagens acontecem. Programar nada mais é que se comunicar com o computador em uma linguagem que, tanto computador, quanto humano entendam. Aprender programação é, portanto, aprender uma nova língua. Na visão de Piaget, as crianças já nascem com a capacidade de aprender, que Papert chama de “aprendizagem piagetiana” ou “aprendizagem sem ensino” (PAPERT, 1985, p. 20). As crianças são capazes de construir suas próprias estruturas intelectuais e, as crianças como construtoras de seu próprio conhecimento, utilizam de materiais e de “metáforas sugeridas pela cultura que a rodeia” para construir tais conhecimentos.

Aprender programação é inverter os papéis: não é mais o computador que diz a você o que fazer, mas é você quem dirá ao computador o que ele deve fazer. É assumir um papel de poder, é ter o poder sob um dos equipamentos tecnológicos mais poderosos da atualidade. O computador pode, muitas vezes, levar a criança a somente reproduzir algum conhecimento, porém, no ambiente LOGO, a criança ensina o computador a pensar e, assim, pensa sobre seu próprio pensamento, fazendo com que a criança desenvolva a habilidade de pensar de forma reflexiva.

2.4 A visão multiparadigmática da programação

Aqui, o multi não é meramente multi, não se trata de uma combinação de paradigmas, de uma aglutinação de características de paradigmas distintos, mas sim de uma visão para além dos paradigmas e de, talvez, uma convergência entre paradigmas.

Alan Kay (1996) acreditava que a POO que ele propunha com Smalltalk, assim como a programação funcional em Lisp, se tratava, na realidade, de uma “cristalização de estilo” ao invés de uma “aglutinação de características” (KAY, 1996, p. 512). Para ele, as técnicas fundamentais que caracterizavam o poder do estilo orientado a objetos – “estado persistente, polimorfismo, instanciação e métodos como metas para o objeto” (KAY, 1996, p. 543) – podiam ser usadas em qualquer linguagem de programação, não requerendo, assim, uma linguagem orientada a objetos para isso. A POO de Alan Kay é, então, um estilo de programação que independe de linguagem de programação. Os paradigmas vistos, portanto,

como estilos de programação podem ser trabalhados sem a presença de uma linguagem de programação específica. O foco é, portanto, nos paradigmas e não nas linguagens.

De certa forma, se faz aqui uma crítica ao que se entende hoje por paradigma de programação e como os paradigmas são trabalhados/abordados. O paradigma imperativo está tão enraizado – e inclusive a forma em que a programação é abordada nos cursos da área contribui para isso – que a programação tem se resumido à programação imperativa. Com exceção da programação orientada a objetos, que ganhou certa popularidade nas últimas décadas, no geral, os demais paradigmas são abordados em uma única disciplina, de forma que os profissionais desconhecem a maioria deles e sequer compreendem a importância de cada um deles. E, mesmo assim, como visto anteriormente, a POO absorveu características da programação imperativa que hoje são tidas como "naturais" dela – como variáveis e operações de atribuição. Contrário ao exposto anteriormente, o resultado tem sido o foco nas linguagens e não nos paradigmas.

Somado a isso, atualmente, aprender uma linguagem de programação não se trata meramente de aprender uma língua nova. Requer, além disso, conhecimentos específicos, técnicos, de computação. Um professor que deseja ensinar a seus alunos conceitos de programação deve, primeiramente, fazer toda uma contextualização técnica e teórica, de modo a levar os alunos a compreenderem o funcionamento de uma máquina ao executar uma instrução de um programa. Nesse sentido, não superamos ainda a barreira da dependência conceitual programa-máquina. E, se desejamos levar à população em geral conhecimentos de programação, é fundamental que programa e máquina sejam tratados de maneira independente, ou teremos, na educação básica, alunos com as mesmas dificuldades de aprendizagem em programação que alunos da graduação. Reforçando a necessidade da independência conceitual entre software e hardware, na mesma visão de Papert, não objetivamos o ensino de técnicas de programação, ou de computação em geral, na educação básica, mas sim a construção pelos alunos de suas próprias estruturas intelectuais, fornecendo-lhes materiais concretos a serem explorados durante esse processo. O ensino da programação como um meio e não como um fim.

CAPÍTULO 3
METODOLOGIA PARA ENSINO DE PROGRAMAÇÃO NA EDUCAÇÃO BÁSICA: uma proposta

Neste capítulo, descreve-se uma metodologia de ensino a ser utilizada com alunos da Educação Básica a fim de propiciar a esses alunos, em contato com uma forma diferente de programação, uma maneira de construir eles mesmos as suas próprias estruturas do conhecimento. Tal metodologia se baseia na filosofia e ambiente LOGO, na computação desplugada, na robótica situada, na programação orientada a objetos (POO) proposta por Alan Kay, no "Joe Book" e em um aplicativo para ensino de programação chamado LightBot[16].

LightBot trata-se de um jogo de quebra-cabeças, cujo objetivo é programar um "robozinho" de modo que, através de um conjunto finito de comandos, ele acenda todas as "luzes" dos blocos azuis de um tabuleiro. A medida em que sobe o nível do jogo, incrementa-se a complexidade do tabuleiro e passa-se a poder utilizar um conjunto de comandos mais sofisticados. Gouws, Bradshaw e Wentworth (2013) desenvolveram o *Computational Thinking Framework* (CTF), que enumera conjuntos de habilidades que compõem o Pensamento Computacional e, a partir do CTF, avaliaram em que grau o LightBot trabalha tais habilidades, em uma escala de 0 a 3. Posteriormente, o resultado nessa escala foi convertido em porcentagem e o LightBot obteve uma média de 74%, tendo se destacado nos conjuntos de habilidades referentes a Processos e Transformações (75%) e Modelos e Abstrações (92%).

Primeiramente, neste trabalho se propõe uma metodologia a ser aplicada sem o uso de qualquer ferramenta computacional, seguindo a ideia da computação desplugada. Utiliza-se essa abordagem para fins de viabilização da aplicação da metodologia, visto que nem sempre se tem à disposição um laboratório de informática ou qualquer outro recurso computacional. Além disso, e talvez o mais relevante, é que busca-se demonstrar que a computação está presente não somente nos computadores e que é possível ensinar princípios de programação sem o uso de ferramentas computacionais ou de uma linguagem de programação específica. Ademais, o objetivo da metodologia não se trata do ensino de uma linguagem ou técnica específica de programação, mas sim de propiciar a alunos o contato com a formalização da resolução de problemas, a elaboração estruturada de uma solução a fim de se atingir um objetivo, a prática de atividades que trabalhem o raciocínio lógico, entre outros.

[16] https://lightbot.com/

De maneira geral, a metodologia trata-se de uma atividade a ser realizada com os alunos, que consiste na construção e realização de um jogo. O jogo é constituído de diversas fases, em que uma fase deve possuir um nível de dificuldade superior ao da fase anterior. Para cada fase, é construído um cenário e é determinada uma tarefa a ser realizada. O cenário construído impõe regras e a tarefa especificada pode ser composta por um ou mais objetivos a serem cumpridos. Uma tarefa é concluída se todos os objetivos da tarefa forem atingidos. Ao se concluir uma tarefa, passa-se, portanto, para uma nova fase, onde um novo cenário é construído e uma nova tarefa é especificada. O jogo pode ter uma curta duração ou possuir diversas fases e se estender ao longo de várias aulas, dependendo dos objetivos e conceitos que o professor deseja trabalhar. Além disso, a solução para a tarefa é dada a partir da interação com objetos presentes no cenário, de modo a se atingir todos os objetivos especificados.

No cenário, portanto, podem ser inseridos objetos, devendo ser inserido ao menos um objeto, em que cada objeto é visto como um objeto da programação orientada a objetos e, na visão de Alan Kay, como um minicomputador com todo o poder computacional de um computador-todo. Assim como todo computador, os alunos podem interagir com os objetos e, para interagir com o objeto, os alunos, primeiramente devem compreender de que maneira se dará essa interação. Assim como na POO proposta por Alan Kay, a interação com objetos se dá a partir da troca de mensagens. Dessa forma, os alunos devem interagir com os objetos enviando mensagens a eles. A cada mensagem recebida, uma ação é desempenhada pelo objeto. Porém, deve-se definir quais mensagens podem ser enviadas aos objetos de uma mesma classe e, para cada mensagem, qual ação é desempenhada pelo objeto ao receber tal mensagem. Em termos de programação, estará especificando-se classes de objetos. Ao se especificar uma classe de objetos Cubo, por exemplo, cubos concretos poderão ser inseridos no cenário e, a partir daí, interações com esses cubos poderão ser realizadas.

Para cada fase do jogo, três etapas básicas devem ser realizadas, que são elas: Etapa 1 – construção do cenário, especificação da tarefa a ser realizada, bem como o(s) objetivo(s) da tarefa, composição de classes de objetos a serem inseridos no cenário e possível inserção de objetos concretos no cenário, a partir das classes de objetos criadas. A Etapa 1 é melhor descrita na seção 3.1; Etapa 2 – trata-se de, a partir da interação com objetos inseridos no cenário, concluir a tarefa dada – novos objetos podem ser inseridos nessa etapa, caso necessário. Nessa etapa, é elaborada, portanto, uma solução formal para a tarefa especificada na etapa anterior. Tal seção é melhor descrita na seção 3.2; Etapa 3 – trata-se da validação da solução elaborada na etapa anterior, através da simulação das interações com o(s) objeto(s), a

fim de verificar se todos os objetivos da tarefa foram devidamente atingidos. Essa etapa pode ser realizada após a conclusão da etapa anterior ou em paralelo com a Etapa 2, o que é melhor descrito nas seções 3.2 e 3.3. A Etapa 3 é descrita na seção 3.3. Ademais, a seção 3.4 traz exemplos de como incrementar a dificuldade das fases.

Por fim, sugere-se que os alunos sejam divididos em dois ou três grupos, em que o primeiro grupo ficará responsável pela execução da Etapa 1, o segundo grupo, pela execução da segunda etapa e a Etapa 3 poderá ser executada por um terceiro grupo ou por um conjunto de alunos do primeiro grupo, em que, para cada objeto no cenário, um aluno deverá ficar responsável por simular as interações com esse objeto. O interessante é que o primeiro e segundo grupos sejam conjuntos de alunos distintos, uma vez que o primeiro grupo ficará responsável pela elaboração do cenário, definição da tarefa a ser cumprida e, de certa forma, pela determinação das regras a serem cumpridas no jogo, enquanto que o segundo grupo ficará responsável por fornecer uma solução adequada para a tarefa dada, seguindo-se tudo o que foi especificado.

3.1 Etapa 1: construindo o cenário do jogo, definindo uma tarefa, especificando classes de objetos e inserindo objetos concretos ao cenário do jogo

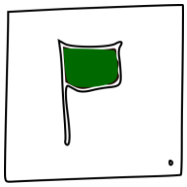
Na primeira etapa, o primeiro grupo deverá construir um cenário para o jogo, definir uma tarefa a ser cumprida, bem como o(s) objetivo(s) da tarefa, e, possivelmente, criar novas classes de objetos e inserir objetos no cenário. Durante todo o jogo, o professor deve mediar a realização, principalmente, dessa primeira etapa, de modo que a atividade cumpra com o(s) objetivo(s) que deseja alcançar, sobretudo nas primeiras fases, uma vez que os alunos ainda não estarão habituados com a metodologia utilizada. No decorrer do jogo, é interessante que o professor leve os alunos a terem contato com novos conceitos de programação a serem trabalhados. Por exemplo, na primeira fase, é interessante o professor propor uma classe de objetos a ser utilizada para inserção de objetos concretos no cenário e interação com esses objetos e, somente nas seguintes, sugerir que os alunos criem suas próprias classes de objetos. Ao trabalhar com a composição, definição, de novas classes, por exemplo, os alunos estarão trabalhando a sua capacidade de abstração.

Um cenário do jogo trata-se de uma espécie de tabuleiro, construído a partir de um conjunto de placas, semelhantes às de um tatame ou de um tapete para crianças, porém, por simplificação, as placas não precisam necessariamente se encaixar uma na outra. As placas podem ser feitas, por exemplo, com EVA, cartolina ou papelão. A princípio, pensou-se que

elas precisam apenas ter o mesmo tamanho e o formato de um quadrado, ou seja, os quatro lados precisam ter a mesma medida. E, como dito anteriormente, a cada nova fase, um novo cenário deverá ser construído, modificando-se a disposição das placas, inserindo-se ou removendo-se placas e/ou inserindo novos objetos ao cenário.

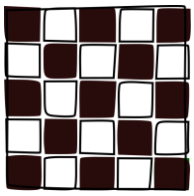
Considerando como uma primeira tarefa fazer com que um objeto, a partir de instruções dadas a ele, saindo de sua posição inicial no cenário, percorra as placas, uma a uma, até chegar a uma determinada placa, duas placas diferentes das demais deverão ser utilizadas: uma correspondente a uma Partida e a outra, a uma Chegada. Nos exemplos aqui ilustrados, conforme Figuras 3 e 4, respectivamente, utilizou-se como Partida uma placa com o desenho de uma bandeira verde e, como Chegada, uma placa quadriculada em preto e branco.

Figura 3 – Partida



Fonte: autora

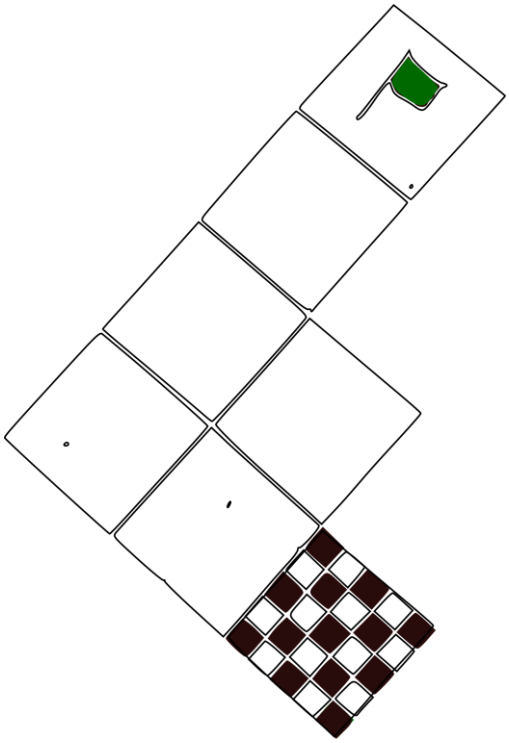
Figura 4 – Chegada



Fonte: autora

O grupo deverá, portanto, para essa fase, dispor as placas, unindo-as umas às outras, lado a lado, de forma a se criar uma espécie de plano cartesiano, ou até mesmo um tatame ou tapete para crianças, de modo que, caminhando-se sobre as placas, uma a uma, exista ao menos um caminho possível da Partida até a Chegada. A figura a seguir ilustra um exemplo de disposição das placas, resultando assim em um cenário do jogo, o qual chamaremos aqui de cenário exemplo:

Figura 5 – Cenário exemplo



Fonte: autora

O cenário impõe regras no sentido que a disposição das placas determina, por exemplo, quais os possíveis caminhos um objeto pode percorrer, quais placas devem ser visitadas ou quais placas devem ser evitadas, etc. O ambiente ao qual o objeto está inserido interfere diretamente nas ações que deverão, ou poderão, ser executadas por ele.

Para a tarefa inicial mencionada, utilizaremos como objeto um cubo. O cubo será, portanto, um objeto concreto da classe Cubo, será inserido no cenário e, durante a Etapa 2, o segundo grupo deverá interagir com o cubo, enviando mensagens a ele, de modo a fazer com que o cubo, estando em posição inicial correspondente à Partida chegue até a Chegada. O conjunto de possíveis mensagens que poderão ser enviadas a um objeto da classe Cubo será descrito mais à frente.

Diante disso, será necessária a confecção de ao menos um cubo. É interessante que o cubo tenha as mesmas dimensões das placas. A seguir é ilustrado o cubo utilizado nos exemplos deste trabalho. Para fins de demonstrações, o cubo exemplo possui uma letra de A a F em cada um dos seus lados.

Figura 6 – Cubo exemplo

Fonte: autora

Nesta metodologia, para “criação” de um objeto concreto da classe Cubo, um nome deve ser dado ao cubo a ser criado. Isso pode ser feito através de um comando: `cubo novo chamado “Patrick”`, em que “Patrick” é o nome dado ao cubo e que será usado para se referir a ele ao longo do jogo. Criar um objeto concreto é, portanto, instanciar um novo objeto, em termos de POO. Além disso, podemos definir que, a partir desse momento, Patrick é inserido “automaticamente” no cenário do jogo em posição inicial correspondente à Partida. E, como mencionado anteriormente, um objeto pode ser inserido no cenário tanto durante a Etapa 1, pelo primeiro grupo, quanto durante a Etapa 2, pelo segundo grupo. Se o objetivo da tarefa for, por exemplo, fazer com que Patrick atinja a Chegada, Patrick já deve estar inserido no cenário ao dar início à Etapa 2, porém se o objetivo for fazer com que um cubo qualquer atinja a Chegada, a criação desse cubo pode ser feita na segunda etapa. Consideraremos para os exemplos seguintes que o objetivo seja levar Patrick até a Chegada.

É importante salientar ainda que, a cada nova fase, um novo cenário é criado, com novos objetos e novos objetivos. Portanto, se, para a fase seguinte, for de interesse a presença do cubo Patrick novamente, um novo cubo Patrick deve ser instanciado. É como se, para cada fase, um novo programa fosse criado.

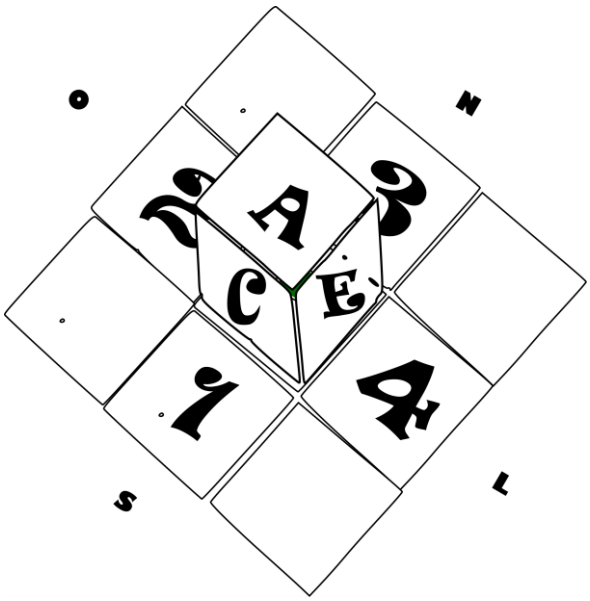
Destaca-se que Patrick é visto, portanto, como um objeto da programação orientada a objetos, como um minicomputador com o mesmo poder computacional de um “computador-todo” e não interessa ao usuário, ao programador, o estado interno do Patrick ou quais processamentos internos ele realiza ao receber uma mensagem. Ao programador interessa apenas como ele deve proceder para interagir com Patrick, quais mensagens ele deve enviar a Patrick para que ele realize determinada ação. Salienta-se ainda que, para a execução da Etapa 2, é necessário, primeiramente, compreender de que maneira pode-se interagir com um objeto da classe Cubo, de modo a fazê-lo cumprir todos os objetivos da fase, o que é descrito a seguir.

Imaginemos que um objeto da classe Cubo nunca se move arrastando-se. Ao invés disso, ele tomba para frente, para trás, para um lado ou para outro. Por exemplo, um cubo pode tombar para frente e, assim, passar a ocupar a placa que estava a sua frente. Além disso, um cubo se move a partir de comandos dados a ele, que, como dito anteriormente, tratam-se de mensagens enviadas ao cubo em questão. Dessa forma, as mensagens seguem um mesmo formato: o objeto a que se deseja enviar uma mensagem (no nosso exemplo, Patrick), seguido de uma vírgula, seguido do restante da mensagem a ser enviada ao objeto. Abaixo, tem-se as mensagens que, inicialmente, podem ser enviadas a um objeto cubo (no exemplo, Patrick), já no formato descrito:

```
Patrick, vá para sul
Patrick, vá para norte
Patrick, vá para leste
Patrick, Vá para oeste
```

A maneira que se deverá proceder com o envio das mensagens para que se alcance o objetivo do jogo será melhor descrita mais à frente, nas seções 3.2 e 3.3, correspondentes às Etapas 2 e 3 do jogo, respectivamente. Os resultados ao se enviar ao cubo Patrick as mensagens descritas acima serão demonstrados a seguir, considerando um cenário inicial, ilustrado pela Figura 7, chamado aqui de cenário de demonstração. Conforme exibido na figura, é interessante que se indique, nesse caso, pelo menos à primeira vez, as direções Norte (N), Sul (S), Leste (L) e Oeste (O) para que se tenha uma referência ao indicar para onde Patrick deve ir.

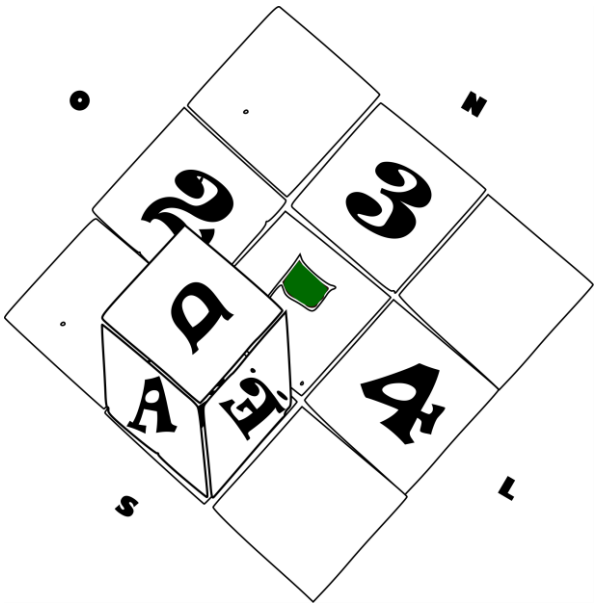
Figura 7 – Cenário inicial de demonstração do envio de mensagens ao cubo Patrick



Fonte: autora

Ao se enviar, por exemplo, a mensagem `Patrick, vá para sul`, Patrick irá receber a mensagem e proceder com uma ação decorrente da mensagem enviada. Logo, o lado C do cubo Patrick passará a tocar a placa 1 do cenário de demonstração. Patrick, portanto, ficará disposto no cenário conforme figura a seguir:

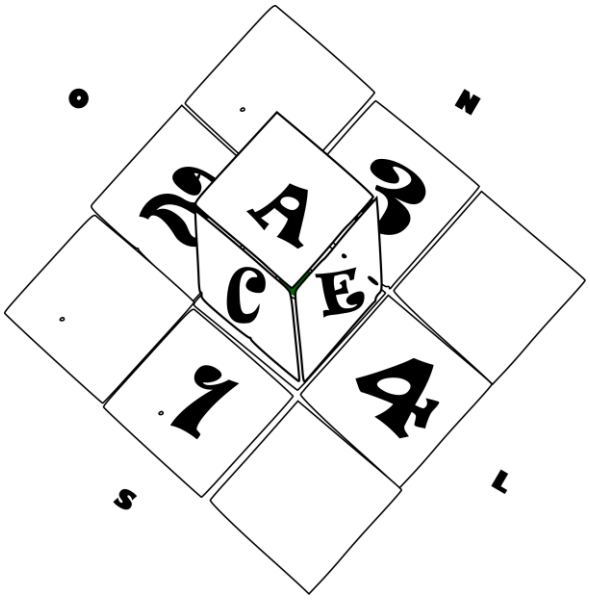
Figura 8 – Cenário de demonstração após envio da mensagem Patrick, vá para sul



Fonte: autora

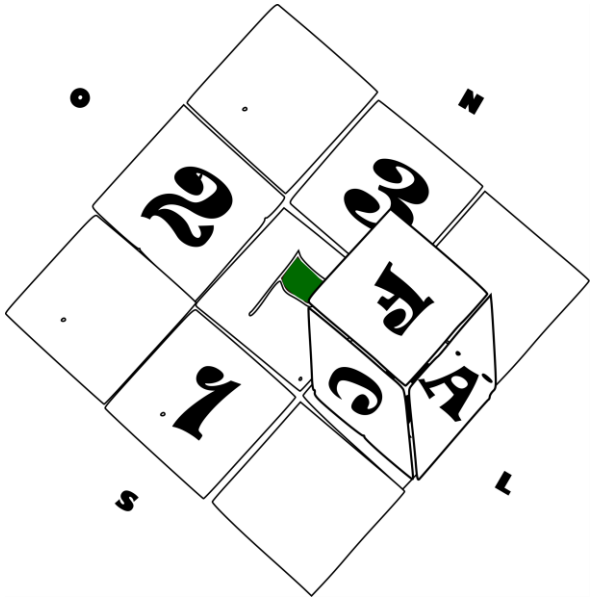
Se, ao invés de enviar a mensagem `Patrick, vá para sul`, enviar-se a mensagem `Patrick, vá para leste`, o lado E do cubo Patrick passará, então, a tocar a placa 4 do cenário de demonstração. Para uma melhor visualização, tem-se a seguir, lado a lado, o cenário de demonstração antes e depois do envio da mensagem, ilustrado nas Figuras 9 e 10, respectivamente.

Figura 9 – Cenário de demonstração inicial



Fonte: autora

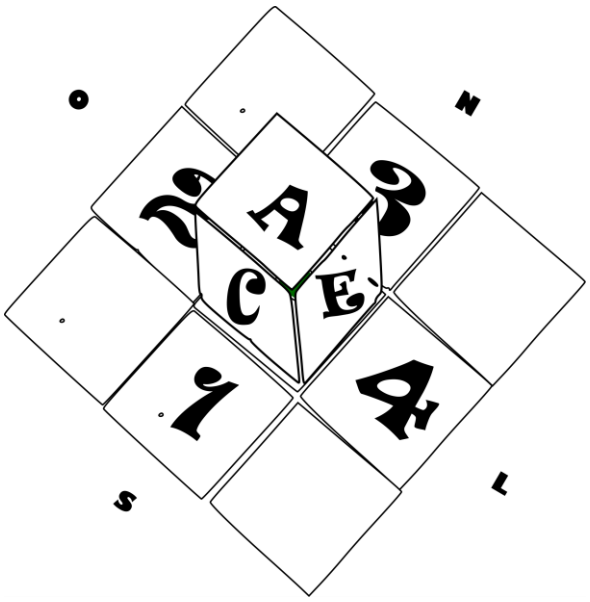
Figura 10 – Cenário de demonstração após o envio da mensagem Patrick, vá para leste



Fonte: autora

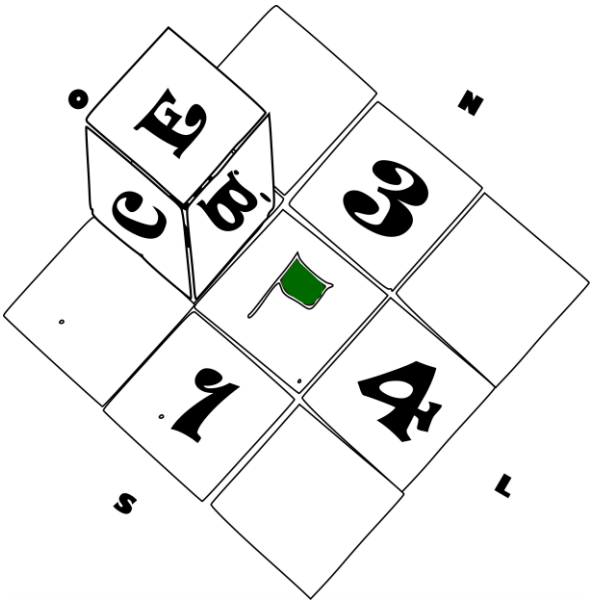
Seguindo o mesmo raciocínio, ao se enviar a Patrick a mensagem `Patrick, vá para oeste`, considerando novamente a posição inicial do cubo, o mesmo irá para a placa 2, ficando com seu lado E voltado para cima, como demonstrado na Figura 12. Novamente, o cenário inicial é colocado ao lado da figura em questão para uma melhor visualização do antes e do depois.

Figura 11 – Cenário inicial de demonstração



Fonte: autora

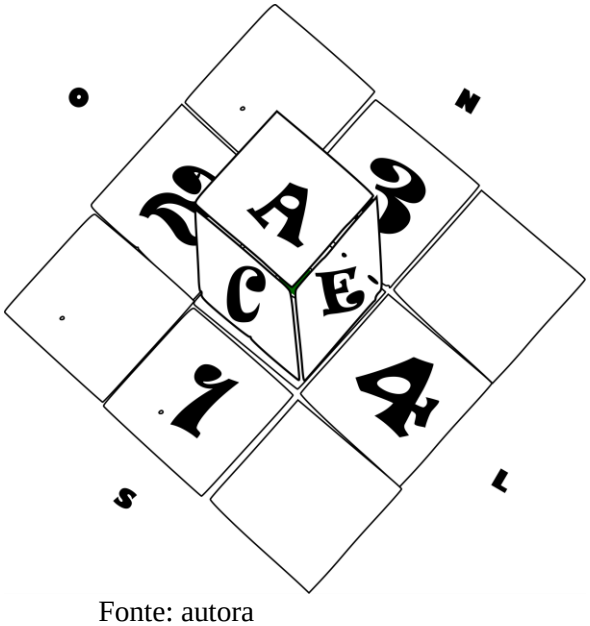
Figura 12 – Cenário de demonstração após envio da mensagem Patrick, vá para oeste



Fonte: autora

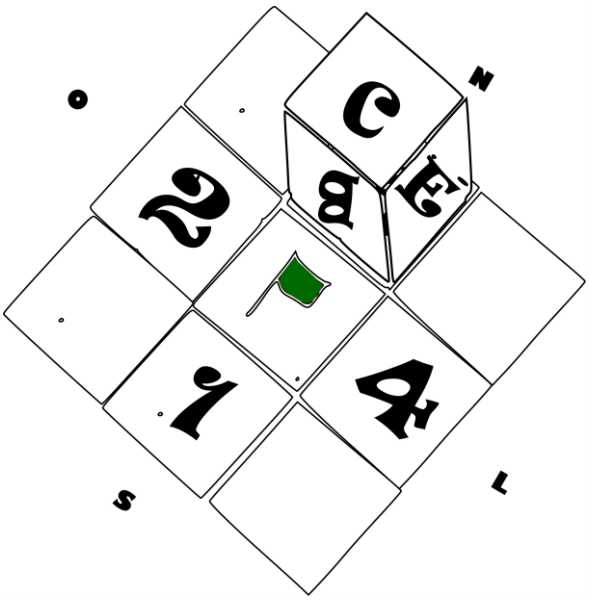
Por fim, ao enviar a mensagem `Patrick, vá para norte`, o cubo se moverá para a placa 3, ficando com seu lado C voltado para cima, conforme demonstrado abaixo:

Figura 13 – Cenário inicial de demonstração



Fonte: autora

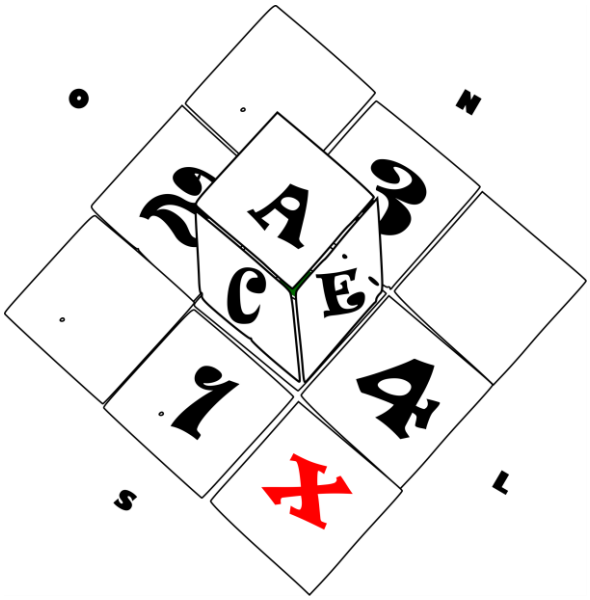
Figura 14 – Cenário de demonstração após o envio da mensagem Patrick, vá para norte



Fonte: autora

Dessa forma, portanto, é importante salientar que, considerando a imagem abaixo, um objeto da nossa classe Cubo nunca poderá se mover, por exemplo, do ponto em que se encontra até a placa X, ou seja, na diagonal, através de um único comando.

Figura 15 – Um cubo não pode ser mover diagonalmente através de um único comando



Fonte: autora

Após o cenário ter sido criado, a tarefa a ser realizada ter sido definida e ter sido explicado como interagir com os objetos e, possivelmente, como inserir novos objetos concretos no cenário, o segundo grupo deverá prosseguir com a realização da Etapa 2, que é descrita na seção a seguir.

3.2 Etapa 2: elaboração da solução

Essa etapa consiste em criar uma solução para a tarefa definida pelo primeiro grupo. Reforçando, trata-se de cumprir todos os objetivos especificados, interagindo-se com os objetos concretos presentes no cenário do jogo, através da troca de mensagens. Os alunos estarão trabalhando, portanto, a habilidade de propor uma solução para determinado problema. Considerando a tarefa inicial descrita na seção anterior – fazer Patrick ir da Partida até a Chegada –, os alunos deverão escrever em uma folha de papel as mensagens que julgarem ser necessárias para que Patrick, estando em posição inicial correspondente à Partida, chegue até a Chegada. Conforme explanado no início do capítulo, essa etapa pode

ocorrer paralelamente à Etapa 3, que consiste na simulação da interação com o objeto, ou seja, em manipular o objeto, conforme mensagens enviadas a ele. Dessa forma, a cada mensagem escrita na folha de papel, a interação correspondente a tal mensagem será imediatamente simulada. Do contrário, todas as mensagens deverão ser escritas na folha de papel para, ao final, suas interações correspondentes serem simuladas. É importante que o professor defina, antes do início do jogo, qual estratégia irá adotar, de acordo com o conceitos que se deseja trabalhar. Por exemplo, adotando a primeira estratégia, o programa criado terá característica da programação funcional, enquanto que a segunda, da programação imperativa. A estratégia pode, inclusive, ser modificada ao longo do jogo, de uma fase para outra.

Considerando o cenário exemplo da Figura 5, abaixo é apresentada, como exemplo, uma das possíveis sequências de mensagens necessárias para que Patrick chegue até a Chegada, considerando as mensagens descritas na seção anterior:

```
Patrick, vá para sul
Patrick, vá para sul
Patrick, vá para leste
Patrick, vá para sul
Patrick, vá para leste
```

Além de trabalharem em conjunto com outros colegas na elaboração de uma solução a um dado problema, nessa etapa, os alunos aprendem a organizar seus pensamentos estrutural e logicamente. Além disso, após a conclusão da Etapa 3, se o(s) objetivo(s) da tarefa não for(em) atingidos, a etapa 2 deve ser realizada novamente. Os alunos devem repensar a solução, fornecer uma nova solução corrigindo o que julgarem necessário. É o “falhar”, no sentido, aqui, de não atingir o objetivo determinado, e tentar novamente quantas vezes forem necessárias até que se atinja tal objetivo. De certa forma, a programação trabalha também a resiliência.

Por fim, nota-se, portanto, que o cenário interfere diretamente nas mensagens a serem enviadas aos objetos, na programação a ser realizada. Ou seja, a programação não é pré-dada, ela emerge da disposição atual do ambiente, da interação do objeto com o ambiente e com outros objetos. A mensagem a ser enviada a um cubo será determinada pela posição atual do cubo em relação ao ambiente e em relação ao que se tem ao seu redor. Trazendo-se as ideias da robótica situada para este contexto, tem-se uma “programação situada”.

A seguir, a Etapa 3, que consiste na validação da solução, é descrita.

3.3 Etapa 3: validação da solução

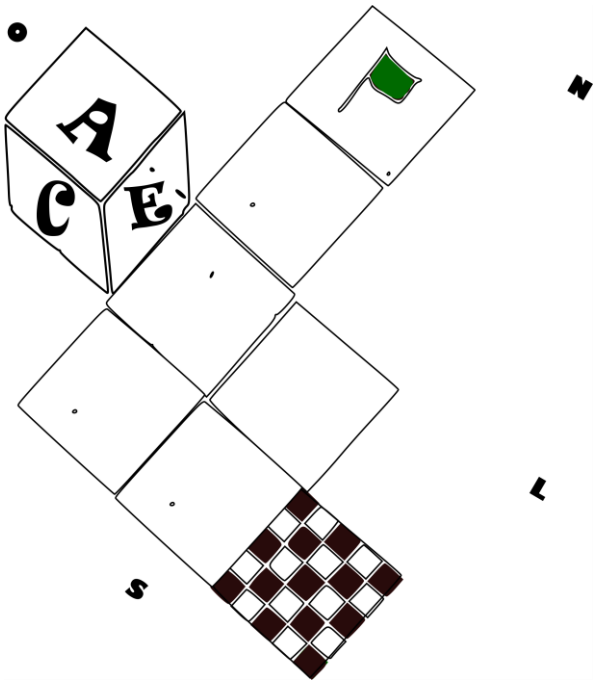
A Etapa 3 do jogo consiste em simular a ações que serão realizadas pelos objetos a partir das mensagens enviadas a eles, na mesma sequência em que foram especificadas na folha de papel, ou seja, considerando a tarefa inicial, manipular Patrick de acordo com o que indica cada mensagem enviada a ele, uma a uma, de modo a verificar se, ao final da sequência de mensagens, atinge-se o objetivo da tarefa. Para que o objetivo seja atingido, portanto, ao final da simulação das ações de Patrick, ele deve estar posicionado no local de Chegada, que aqui chamaremos de objetivo principal – conforme será descrito na seção 3.4, outros objetivos podem ser especificados, de maneira a incrementar o nível de dificuldade do jogo. O jogo não deve prosseguir em duas situações: i. caso alguma regra do jogo seja violada, como, por exemplo, enviar uma mensagem não definida para a classe de objetos ou, ao simular a ação de um objeto, resultar no deslocamento do objeto para fora do cenário; ii. caso, ao final da simulação, os objetos não sejam cumpridos, que, no exemplo, corresponde a Patrick, não atingir a Chegada.

Por exemplo, considerando o cenário exemplo da Figura 5, imaginemos que o grupo de alunos, tenha especificado a seguinte sequência de mensagens:

```
Patrick, vá para sul
Patrick, vá para sul
Patrick, vá para oeste
Patrick, vá para sul
Patrick, vá para leste
```

Após a simulação da ação decorrente do envio da terceira mensagem – `Patrick, vá para oeste` – o resultado será conforme ilustrado na Figura 16, ou seja, Patrick sairá do cenário do jogo. Estará, portanto, em um local onde não há placas.

Figura 16 – Exemplo: cubo fora do cenário

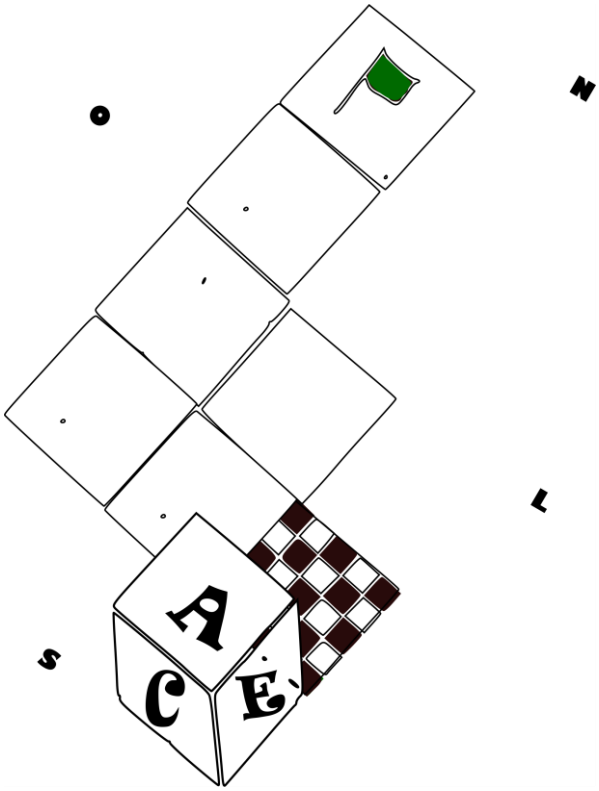


Fonte: autora

Um outro exemplo seria: considerando a sequência de mensagens abaixo, ao simular as ações decorrentes, apesar de Patrick, após a simulação da penúltima mensagem, chegar até à Chegada, ao prosseguir com a simulação da última mensagem, o objeto cubo finalizará em um local diferente da Chegada, conforme ilustrado pela Figura 17.

```
Patrick, vá para sul
Patrick, vá para sul
Patrick, vá para leste
Patrick, vá para sul
Patrick, vá para leste
Patrick, vá para sul
```

Figura 17 – Exemplo: cubo finaliza em local distinto da Chegada



Fonte: autora

Caso ocorra alguma das duas situações mencionadas anteriormente, o jogo deve prosseguir de uma das duas seguintes maneiras, o que deve ser definido antes do início do jogo: i. o objeto deve ser retornado ao seu local de Partida, de modo que possa ser simulada a sequência de comandos desde o início novamente, o segundo grupo deve, então proceder com as correções que julgar necessárias e, posteriormente, prosseguir com a validação da solução; ii. retornar ao passo anterior, efetuar as correções e prosseguir dali.

Alcançando-se o objetivo da tarefa, o jogo poderá prosseguir para a próxima fase. O grupo 1 deverá, portanto, criar um novo cenário, ou seja, realizar a Etapa 1 novamente, porém agora procurando aumentar o nível de dificuldade do jogo em relação à fase anterior. Na seção seguinte, são descritos alguns exemplos de como se aumentar a dificuldade das fases.

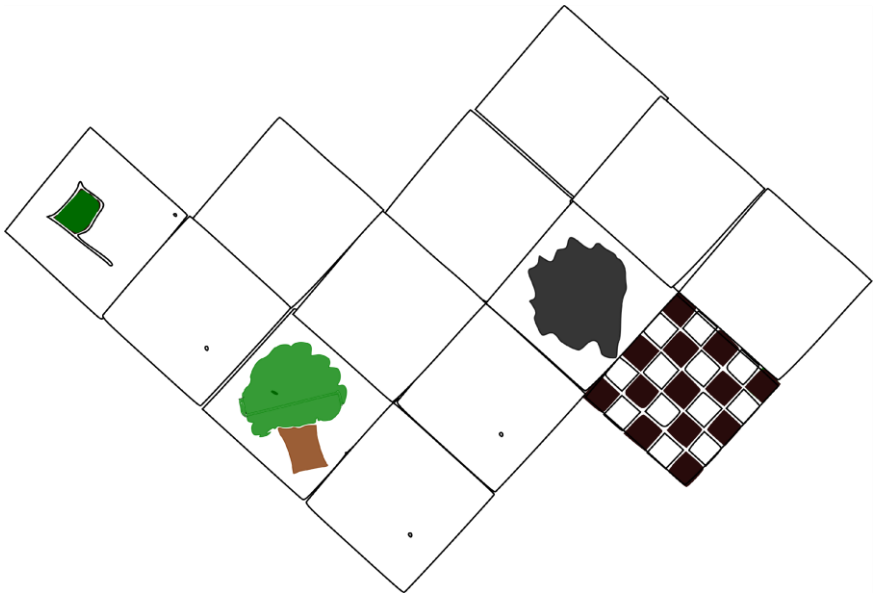
3.4 Aumentando o nível de dificuldade de uma fase

Esta seção traz ideias de recursos que podem ser utilizados a fim de se aumentar o nível de dificuldade do jogo. O que aqui se sugere é que os alunos do primeiro grupo tenham uma certa autonomia para definir estratégias a serem utilizadas, de modo que sejam trabalhadas habilidades, como, por exemplo, a criatividade. Porém, é interessante também que o professor faça intervenções, de acordo com os objetivos pedagógicos que almeja, com os conceitos que deseja trabalhar.

Aqui, pensa-se em alguns fatores que podem aumentar a dificuldade do jogo, que podem ser considerados em conjunto ou não: i. aumentar a quantidade de placas do cenário; ii. dispor as placas de maneira que se aumente a quantidade de passos necessários para se alcançar os objetivos; iii. adicionar novas mensagens que possam ser enviadas a objetos de uma classe; iv. criar outras classes de objetos, de modo que um objeto possa interagir com os outros e/ou permitir a criação de vários objetos de uma mesma classe; v. utilizar outros recursos/conceitos da programação a serem trabalhados, como, por exemplo, comandos de seleção, recursão, procedimentos, etc.

Uma outra maneira de se incrementar a dificuldade do jogo é criando novas regras ou novos objetivos. Por exemplo, pode-se inserir “bloqueios” no cenário, de forma a determinar que se faça um desvio a esses bloqueios, conforme ilustrado na imagem a seguir:

Figura 18 – Cenário com bloqueios

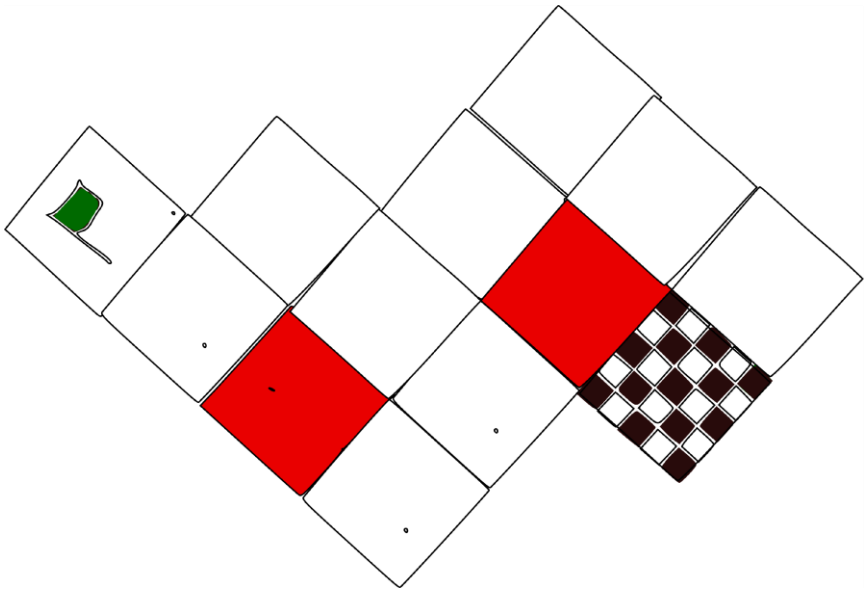


Fonte: autora

No exemplo da Figura 18, o cenário contém uma árvore e um “buraco negro” que devem ser desviados pelo cubo. Um cubo, portanto, não poderá passar pelas placas que

contém esses bloqueios. Bloqueios podem também ser simulados inserindo-se, por exemplo, placas de cores diferentes, indicando que o cubo não deve passar por essas placas. A figura a seguir ilustra esse outro exemplo, em que foram inseridas placas vermelhas, indicando que um cubo não deve passar por essas placas.

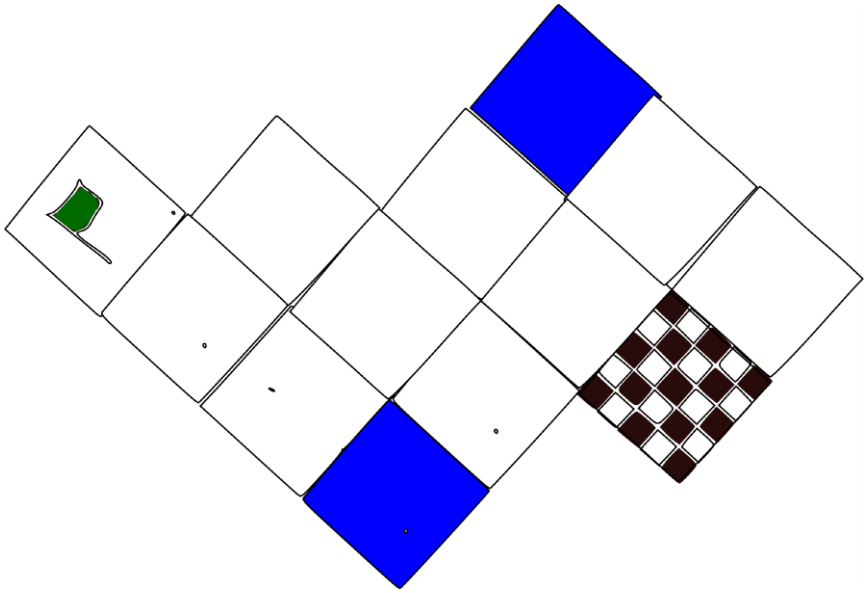
Figura 19 – Criando bloqueios com placas de cores distintas



Fonte: autora

Uma outra possibilidade seria usar uma cor diferente para indicar que o objeto cubo, por exemplo, deve passar pelas placas daquela cor. No exemplo a seguir, foram utilizadas duas placas de cor azul para indicar que o cubo deve passar por aquelas placas.

Figura 20 – Definindo placas em que o cubo deve passar



Fonte: autora

Pode-se definir também que, considerando que um cubo, ao ser criado, inicie com o lado A para cima, ele deve finalizar com o lado E, por exemplo, voltado para o sul. Ou pode-se pensar em uma fase mais dinâmica em que o cenário conterá vários objetos e cada aluno deverá ser responsável por determinar as mensagens a serem enviadas a um objeto específico e, assim, todos devem trabalhar em conjunto na elaboração de uma solução para atingir objetivos específicos. Enfim, as possibilidades são diversas e vale usar a imaginação.

Por último, destaca-se que, seguindo as ideias de Seymour Papert, durante todo o processo os alunos têm papel ativo no processo de aprendizagem, trabalham ativamente na construção de suas próprias estruturas do conhecimento, cabendo ao professor apenas mediar e orientar a “caminhada”.

CONSIDERAÇÕES FINAIS

Com a popularização das tecnologias digitais e o surgimento do termo “Pensamento Computacional”, em 2006, que compreende um conjunto de habilidades que pessoas em pleno século XXI devam ter e não somente cientistas da computação, reacende e aquece a discussão acerca do ensino de conceitos computacionais à população em geral e cresce consideravelmente o número de pesquisas sobre o tema na segunda década dos anos 2000. Enquanto práticas como o ensino de programação são adotadas a nível nacional em muitos países, no Brasil são poucas as iniciativas e faltam estudos que visem a qualidade e profundidade do processo de ensino-aprendizagem. Com isso, neste trabalho, buscou-se elaborar uma proposta para o ensino de programação na Educação Básica.

Nos primeiros contatos com a temática, algo chama a atenção: uma das justificativas para o ensino da programação na educação básica é que aprender programação faz com que os alunos adquiram habilidades como raciocínio lógico, porém, por outro lado, as disciplinas de programação nos cursos da área apresentam altos índices de evasão e reprovação e um dos motivos apontados por autores é justamente a falta nos alunos de habilidades que deveriam ter sido adquiridas durante o ensino regular, tais como o raciocínio lógico. Como então aprender programação ajuda no desenvolvimento do raciocínio lógico se programar requer raciocínio lógico desenvolvido?

As ideias de Papert e da filosofia LOGO podem ser usadas para responder a tal questão. Talvez o equívoco da literatura esteja em dar ênfase ao “ensino da programação”, ou ao “ensino da computação”. O objetivo não se trata de ensinar, a alunos da Educação Básica, programação, ou de ensinar computação, mas sim de fornecer aos alunos materiais concretos para a construção de suas próprias estruturas intelectuais, de propiciar aos alunos um ensino voltado para a aprendizagem diferente da habitual, a aprendizagem por meio da programação. Trata-se de aprender a pensar estruturalmente, a pensar criticamente, a pensar sobre o seu próprio pensamento, a ter resiliência, a trabalhar em equipe, a fornecer uma solução a um dado problema, entre outros, tudo isso por meio da programação, tudo isso programando-se um “cubo programável” para que ele atinja determinado objetivo. E isso requer romper com paradigmas adotados no ensino da programação, requer romper com as barreiras da dependência conceitual entre máquina e programa, requer que profissionais e pesquisadores saiam da “bolha computacional técnica” e busquem objetivos pedagógicos, busquem a qualidade dos processos de ensino-aprendizagem.

A proposta que aqui se faz, portanto, é uma metodologia de ensino em que, primeiramente, não requer o uso de qualquer ferramenta computacional, visando romper com as barreiras econômicas e com as falsas interpretações do que é a computação. Além disso, a metodologia não se trata do ensino de uma linguagem ou técnicas de programação específicas. Ao invés disso, busca-se, através de um jogo, propiciar aos alunos o contato com a elaboração formal de uma solução para um dado problema, com o ato de trabalhar em equipe, de pensar estrutural e logicamente, de errar e tentar novamente até atingir determinado objetivo, enfim, com o desenvolvimento de tantas habilidades que o ato de programar possibilita. No mais, o que se propõe é colocar em prática as ideias de Papert, tornando as crianças ativas no processo de ensino-aprendizagem e utilizando a programação como um meio e não como um fim, e as ideias de Alan Kay, principalmente no que concerne a ruptura conceitual entre máquina e programa. Os programadores, ao programar, não deveriam ter que se preocupar com detalhes de hardware, principalmente se tratando de alunos da Educação Básica, em que o objetivo não é formar programadores, mas sim propiciar o desenvolvimento de habilidades necessárias a qualquer cidadão.

Por último, entende-se que esta seja apenas uma pesquisa inicial, seja apenas um “pontapé” inicial, de tantas outras pesquisas que poderão vir com o intuito de avaliar e aprimorar a metodologia utilizada e difundi-la. Uma das melhorias que pretende-se fazer é acrescentar à metodologia mecanismos para se trabalhar com os alunos a elaboração de procedimentos e o uso de laços de repetição.

REFERÊNCIAS

ANACLETO, Aline Ariana Alcântara; BOENO, Rosangela Maria; ALBERTON, Adriana. Defasagens do aprendizado dos estudantes de Ensino Superior nas áreas Exatas. In: CONGRESSO DE CIÊNCIA E TECNOLOGIA DA UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR, 2012, Dois Vizinhos-PR. **Anais eletrônicos...** Curitiba, UTFPR, 2012. Disponível em: http://revistas.utfpr.edu.br/dv/index.php/CCT_DV/article/viewFile/1038/597. Acesso em: 11 set. 2018.

ARAUJO, Ana Liz Souto Oliveira de; ANDRADE, Wilkerson L.; GUERRERO, Dalton D. Serey. Um mapeamento sistemático sobre a avaliação do pensamento computacional no Brasil. In: **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**. 2016. p. 1147.

BARBOSA, Jorge Luis Victoria. **Holoparadigma: um modelo multiparadigma orientado ao desenvolvimento de software distribuído**. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

BELL, Tim; WITTEN, Ian H.; FELLOWS, Mike. **Computer Science Unplugged**: Off-line activities and games for all ages. Computer Science Unplugged, 1998.

BLATT, Lucas; BECKER, Valdecir; FERREIRA, Alexandre Magno e Silva. Mapeamento Sistemático sobre Metodologias e Ferramentas de apoio para o Ensino de Programação. In: **Anais do Workshop de Informática na Escola**. 2017. p. 815.

BORDINI, Adriana et al. Computação na educação básica no Brasil: o estado da arte. **Revista de Informática Teórica e Aplicada**, v. 23, n. 2, p. 210-238, 2016.

BORDINI, Adriana et al. Pensamento Computacional nos Ensinos Fundamental e Médio: uma revisão sistemática. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. 2017. p. 123.

BUDD, Timothy. **Multiparadigm programming in Leda**. Reading, MA: Addison-Wesley, 1995.

FERRACIOLI, Laércio. Aspectos da construção do conhecimento e da aprendizagem na obra de Piaget. **Caderno Brasileiro de Ensino de Física**, v. 16, n. 2, p. 180-194, 1999.

FERRI, Juliana; ROSA, Selma dos Santos. Como o Ensino de Programação de Computadores Pode Contribuir Com a Construção de Conhecimento na Educação Básica Uma Revisão Sistemática da Literatura. **RENOTE**, v. 14, n. 2, 2016.

FONSECA FILHO, Cléuzio. **História da computação**: o caminho do pensamento e da tecnologia. EDIPUCRS, 2007.

GARCIA, Rogério Eduardo; CORREIA, Ronaldo Celso Messias; SHIMABUKURO, Milton Hirokazu. Ensino de Lógica de Programação e Estruturas de Dados para Alunos do Ensino

Médio. In: **XVII WEI–Workshop sobre o Ensino de Computação**. Belém do Pará–PA. 2008. p. 246-249.

GOUWS, Lindsey Ann; BRADSHAW, Karen; WENTWORTH, Peter. Computational thinking in educational activities: an evaluation of the educational game light-bot. In: **Proceedings of the 18th ACM conference on Innovation and technology in computer science education**. ACM, 2013. p. 10-15.

GUDWIN, Ricardo Ribeiro. Novas fronteiras na inteligência Artificial e na robótica. In: **4º Congresso Temático de Dinâmica, Controle e Aplicações**. UNESP. 2005.

HAILPERN, Brent. Guest editor's introduction multiparadigm languages and environments. **IEEE Software**, v. 3, n. 1, p. 6-9, 1986.

HAILPERN, Brent. Multiparadigm Research: a survey of nine projects. **IEEE Software**, v. 3, n. 1, p. 70-77, 1986.

HASELAGER, Pim. Forma, função e a matéria da experiência. In: QUEIROZ, João; LOULA, Ângelo; GUDWIN, Ricardo. (Org.). **Computação, cognição, semiose**. SciELO-EDUFBA, 2007. P. 251-266.

HONORATO, Renata. Aprender a ler, calcular e... programar: o novo desafio nas escolas. **Veja**, Editora Abril, 14 dez. 2013. Disponível em: http://veja.abril.com.br/tecnologia/aprender-a-ler-calcular-e-programar-o-novo-desafio-nas-escolas/. Acesso em: 11 set. 2018.

KAY, Alan C. A personal computer for children of all ages. In: **Proceedings of the ACM annual conference-Volume 1**. ACM, 1972. p. 1.

KAY, Alan C. The early history of Smalltalk. In: **History of programming languages**. ACM, 1996. p. 511-598.

LENZ, Rosiana Karine; CAMBRAIA, Adão Caron. Ensino de programação no Ensino Fundamental. In: CONGRESSO INTERNACIONAL DE EDUCAÇÃO POPULAR, XV, 2015, Santa Maria. **Anais...** Santa Maria: SIEP, 2015.

LYE, Sze Yee; KOH, Joyce Hwee Ling. Review on teaching and learning of computational thinking through programming: What is next for K-12?. **Computers in Human Behavior**, v. 41, p. 51-61, 2014.

MARQUES, Diego Lopes et al. Atraindo alunos do ensino médio para a computação: Uma Experiência Prática de Introdução à Programação utilizando Jogos e Python. In: **Anais do Workshop de Informática na Escola**. 2011. p. 1138-1147.

MASON, Raina; COOPER, Graham. Introductory Programming Courses in Australia and New Zealand in 2013-trends and reasons. In: **Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148**. Australian Computer Society, Inc., 2014. p. 139-147.

MORAIS, Anuar Daian de. **O desenvolvimento do raciocínio condicional a partir do uso de teste no Squeak Etoys**. Tese (Doutorado em Informática na Educação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2016.

MOTA, Fernanda P. et al. Desenvolvendo o Raciocínio Lógico no Ensino Médio: uma proposta utilizando a ferramenta Scratch. In: **Simpósio Brasileiro de Informática na Educação (SBIE)**. 2014. p. 377.

MÜLLER, Martin; MÜLLER, Tobias; VAN ROY, Peter. Multiparadigm programming in Oz. In: **Workshop on the Future of Logic Programming, International Logic Programming Symposium (ILPS 95)**. 1995.

OLIVEIRA, Millena Lauyse Silva de et al. Ensino de lógica de programação no ensino fundamental utilizando o Scratch: um relato de experiência. In: **XXXIV Congresso da SBC-XXII Workshop de Ensino de Computação**, Brasília. 2014.

PAPERT, Seymour. **Logo**: computadores e educação. Trad. José Arnaldo Valente, Beatriz Bitelman e Afira Vianna Ripper. Editora Brasiliense, 1985.

PEREIRA, Débora Fernandes; BEZERRA JÚNIOR, Elias Vidal. O uso da linguagem Scratch no desenvolvimento do raciocínio lógico. In: JORNADA DE INICIAÇÃO CIENTÍFICA E EXTENSÃO (JICE) DO INSTITUTO FEDERAL DO TOCANTINS, 5., 2014. **Anais...** Tocantins: IFTO, 2014.

PEREIRA, Leonardo. Escolas defendem ensino de programação a crianças e adolescentes. **Olhar Digital**, 6 jun. 2013. Disponível em: http://olhardigital.uol.com.br/noticia/escolas-defendem-ensino-de-programacao-a-criancas-e-adolescentes/35075. Acesso em 11 set. 2018.

PEREIRA JÚNIOR, José Carlos Rocha et al. Ensino de algoritmos e programação: uma experiência no nível médio. In: **XIII Workshop de Educação em Computação (WEI'2005)**. São Leopoldo, RS, Brasil. 2005.

PRIOLLI, Gabriel; RAMOS, Eduardo. **Seymour Papert e Paulo Freire**: uma conversa sobre informática, ensino e aprendizagem. O Futuro da Escola, 1995.

ROBBINS, Philip; AYDEDE, Murat (Ed.). **The Cambridge handbook of situated cognition**. Cambridge University Press, 2008.

RODRIGUES, Rivanilson da Silva. **Ensino de algoritmos e linguagem de programação no nível médio: um relato de experiência**. Trabalho de Conclusão de Curso (Graduação em Licenciatura Plena em Computação) – Universidade Estadual da Paraíba, Patos, 2013.

SANTOS, Gustavo et al. Proposta de atividade para o quinto ano do ensino fundamental: Algoritmos Desplugados. In: **Anais do Workshop de Informática na Escola**. 2015. p. 246.

SANTOS, Priscila S. C.; ARAUJO, Luis Gustavo; BITTENCOURT, Roberto. A Mapping Study of Computational Thinking and Programming in Brazilian K-12 Education. In: **48th Annual Frontiers In Education Conference**. San Jose, California. 2018.

SEBESTA, Robert W. **Conceitos de linguagens de programação**. 9. ed. Porto Alegre: Bookman, 2011.

SILVA, Thiago Reis da et al. Ensino-aprendizagem de programação: uma revisão sistemática da literatura. **Revista Brasileira de Informática na Educação**, v. 23, n. 1, 2015.

SILVA, Victor do Nascimento; NASCIMENTO, Michelle Nery. Investigação da melhoria do aprendizado de alunos do ensino médio da rede pública de ensino através do uso de programação, robótica e jogos digitais. In: **SBGames-Brazilian Symposium of Games and Digital Entertainment**. 2012.

SMITH, Brian Cantwell. The foundations of computing. **Computationalism: new directions**, p. 23-58, 2002.

SOARES, Alexandre Henrique Vieira; BORGES, Henrique Elias; SANTOS, Bruno André. Arquitetura para criação de robôs autônomos emocionais-cognitivos numa perspectiva situada. In: **X Encontro de Modelagem Computacional**. 2007.

TUCKER, Allen. A Model Curriculum for K--12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee. 2003.

TUCKER, Allen B.; NOONAN, Robert E. **Linguagens de programação**: princípios e paradigmas. 2. ed. São Paulo: AMGH, 2008.

VAN ROY, Peter; HARIDI, Seif. Teaching programming broadly and deeply: the kernel language approach. In: **Informatics Curricula and Teaching Methods**, p. 53-62, 2003.

VARELA, Aida Varela; BARBOSA, Marilene Lobo Abreu. Aplicação de teorias cognitivas no tratamento da informação. **Revista Brasileira de Biblioteconomia e Documentação**, v. 3, n. 2, p. 116-128, 2007.

VENÂNCIO, Ludmila Salomão; BORGES, Mônica Erichsen Nassif. Cognição situada: fundamentos e relações com a ciência da informação. **Encontros Bibli: revista eletrônica de biblioteconomia e ciência da informação**, v. 11, n. 22, p. 30-37, 2006.

VUJOŠEVIĆ-JANIČIĆ, Milena; TOŠIĆ, Dušan. The role of programming paradigms in the first programming courses. **The teaching of Mathematics**, n. 21, p. 63-83, 2008.

WAZLAWICK, Raul Sidnei. **História da Computação**. Elsevier Brasil, 2017.

WING, Jeannette M. Computational thinking. **Communications of the ACM**, v. 49, n. 3, p. 33-35, 2006.

ZANATTA, Andrei Cardoso. **Programação de computadores para crianças: metodologia do CODE CLUB Brasil**. Trabalho de Conclusão de Curso (Graduação em Tecnologias da Informação e Comunicação) – Universidade Federal de Santa Catarina, Araranguá, 2015.