UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE EDUCAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIAS
EDUCACIONAIS EM REDE – MESTRADO PROFISSIONAL

Cleitom José Richter

ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS NA EDUCAÇÃO PROFISSIONAL POR MEIO DO DESENVOLVIMENTO DE JOGOS APOIADO PELO AMBIENTE GREENFOOT

Santa Maria, RS
2018

Cleitom José Richter

ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS NA EDUCAÇÃO PROFISSIONAL POR MEIO DO DESENVOLVIMENTO DE JOGOS APOIADO PELO AMBIENTE GREENFOOT

Dissertação apresentada ao Curso de curso de Mestrado Profissional em Tecnologias educacionais em rede, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Tecnologias Educacionais em Rede**

Orientadora: Profª. Drª. Giliane Bernardi

Santa Maria, RS, Brasil
2018

Richter, Cleitom José
ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS NA EDUCAÇÃO PROFISSIONAL POR MEIO DO DESENVOLVIMENTO DE JOGOS APOIADO PELO AMBIENTE GREENFOOT / Cleitom José Richter.- 2018.
217 p.; 30 cm

Orientadora: Giliane Bernardi
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Educação, Programa de Pós-Graduação em Tecnologias Educacionais em Rede, RS, 2018

1. Ensino da computação 2. Desenvolvimento de jogos 3. Programação Orientada a Objetos I. Bernardi, Giliane II. Título.

Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

E-mail: cleitom.richter@iffarroupilha.edu.br

Cleitom José Richter

ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS NA EDUCAÇÃO PROFISSIONAL POR MEIO DO DESENVOLVIMENTO DE JOGOS APOIADO PELO AMBIENTE GREENFOOT

Dissertação apresentada ao Curso de Pós-Graduação Profissional em Tecnologias educacionais em rede, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Tecnologias Educacionais em Rede.**

Aprovado em 27 de agosto de 2018:

_____
Giliane Bernardi, Dra. (UFSM)
(Presidente/Orientadora)

_____
Andre Zanki Cordenonsi, Dr. (UFSM)

_____
Adão Caron Cambraia, Dr. (IFFAR)

Santa Maria, RS
2018

AGRADECIMENTOS

À família por ter sido a base que me possibilita crescer.

Aos estimados colegas/amigos/conselheiros que labutam cotidianamente comigo no IFFAR *Campus* Santo Augusto, pelo incentivo e colaborações que possibilitaram a realização desse trabalho.

Aos queridos mestres e colegas do PPGTER pelas valiosas trocas de experiências e aprendizados produzidos durante as aulas.

Aos estudantes do 3º INFO (Turmas 2017 e 2018) que participaram da pesquisa, pelo empenho e dedicação nas atividades desenvolvidas.

Aos professores da banca, pelas contribuições que muito qualificaram este trabalho.

À minha orientadora, professora Giliane, pelo seu profissionalismo e atenção dispensada às nossas conversas, o que tornou possível a construção apresentada nesse trabalho.

O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.

(José de Alencar)

RESUMO

ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS NA EDUCAÇÃO PROFISSIONAL POR MEIO DO DESENVOLVIMENTO DE JOGOS APOIADO PELO AMBIENTE GREENFOOT

AUTOR: Cleitom José Richter
ORIENTADORA: Drª. Giliane Bernardi

Esta dissertação está associada à linha de pesquisa Desenvolvimento em Tecnologias Educacionais em Rede, do Programa de Pós-Graduação em Tecnologias Educacionais em Rede da Universidade Federal de Santa Maria, e descreve uma pesquisa-ação sobre o ensino de Programação Orientada a Objetos (POO), sob uma perspectiva de desenvolvimento de jogos. A motivação para esta proposta tem origem na constatação de que, diferente do que se possa imaginar, o uso das tecnologias não faz dos jovens da atualidade "experts" em computação, pois, conforme estudos realizados por Souza, Batista e Barbosa (2016), o ensino e a aprendizagem de programação são considerados por muitos professores e alunos um processo difícil. Por esse motivo, as disciplinas que tratam do ensino de programação de computadores nos cursos de computação sofrem com as dificuldades dos estudantes. Tais problemas também são verificados no curso Técnico em Informática Integrado ao Ensino Médio do IFFAR Campus Santo Augusto, pois estudantes apresentam grandes dificuldades em compreender conceitos de programação, sobretudo os que envolvem POO. Nesse sentido, a presente pesquisa buscou compreender como o desenvolvimento de jogos pode contribuir e facilitar a apropriação dos conceitos de POO. Tal investigação ocorreu a partir da criação de protótipos de jogos com apoio do ambiente de desenvolvimento Greenfoot. A temática proposta para esse estudo surge da intenção de se utilizar a motivação encontrada nos jogos para romper as dificuldades de ensino e de aprendizagem de programação orientada a objetos. Trata-se de uma pesquisa aplicada com abordagem quali-quantitativa que objetivou gerar conhecimentos que resultarão em reflexões sobre a prática profissional e para aperfeiçoar o ensino de programação. Por se tratar de um método que envolve interação entre o pesquisador e as pessoas implicadas na situação investigada, por ter caráter esclarecedor do problema, frente ao acompanhamento constante das ações, este estudo utilizou procedimentos técnicos característicos da pesquisa-ação. Para esta pesquisa foram selecionadas duas turmas do curso Técnico em Informática Integrado ao Ensino Médio (3º ano) do IFFAR *Campus* Santo Augusto, sendo cada turma composta por 20 a 26 estudantes. As ações dessa proposta foram desenvolvidas na disciplina de Linguagem de Programação III sendo que, em ambas as turmas, a disciplina citada foi ministrada pelo mesmo professor pesquisador. O desenvolvimento da proposta foi norteado pela estruturação de Unidades de Estudo, abordando o desenvolvimento de protótipos de jogos com a finalidade intrínseca de proporcionar condições para a aprendizagem dos conceitos de orientação a objetos, bem como promover melhoria nas intervenções didáticas no ensino de programação de computadores. A partir da pesquisa, pôde-se perceber boa aceitação dos estudantes à proposta, conforme opiniões coletadas em questionário respondido pelos participantes da pesquisa. Também foi possível verificar que houve evolução na aprendizagem dos alunos, comprovada por testes estatísticos executados com comparativos dos resultados da avaliação realizada por meio do desenvolvimento de mapas conceituais pré e pós-intervenções. Produziu-se também um guia de orientações para ser utilizado por profissionais da área para o ensino de POO.

Palavras-Chave: Ensino da computação**.** Programação Orientada a Objetos. Desenvolvimento de jogos.

ABSTRACT

TEACHING OBJECT ORIENTED PROGRAMMING IN PROFESSIONAL EDUCATION BY GAMING DEVELOPMENT SUPPORTED BY THE GREENFOOT ENVIRONMENT

AUTHOR: CLEITOM JOSÉ RICHTER
ADVISOR: DRª. GILIANE BERNARDI

This dissertation is associated with the research line Development in Network Educational Technologies of the Graduate Program in Educational Network Technologies of the Universidade Federal de Santa Maria (UFSM) and describes an action research on the teaching of Object Oriented Programming (OOP), from a game development perspective. The motivation for this proposal stems from the fact that, unlike what can be imagined, the use of technologies does not make today's young people "experts" in computing, since, according to studies by Souza, Batista and Barbosa (2016), teaching and learning programming are considered by many teachers and students a difficult process. For this reason, the disciplines that deal with the teaching of computer programming in computer courses suffer from the difficulties of the students. These problems are also verified in the IFFAR Campus Santo Augusto Technical Computer Science Course, since students present great difficulties in understanding programming concepts, especially those involving OOP. In this sense, the present research sought to understand how the development of games can contribute and facilitate the appropriation of the OOP concepts. Such research came from the creation of game prototypes with support of the Greenfoot development environment. The proposed theme for this study arises from the intention to use the motivation found in games to overcome the difficulties of teaching and learning object oriented programming. It is an applied research with a qualitative-quantitative approach that aims to generate knowledge that will result in reflections on professional practice and to improve programming teaching. For this research, two classes were selected from the IFFAR *Campus* Santo Augusto course in Computer Science, Integrated in High School (3rd year), with each class consisting of 20 to 26 students. The actions of this proposal were developed in the discipline of Programming Language III and, in both classes, the mentioned discipline was given by the same research professor. The development of the proposal was guided by the structuring of Study Units, addressing the development of game prototypes with the intrinsic purpose of providing conditions for learning object orientation concepts, as well as promoting improvement in didactic interventions in the teaching of computer programming. From the research, it was possible to perceive good acceptance of the students to the proposal, according to the opinions collected in a questionnaire answered by the participants of the research. It was also possible to verify that there was evolution in student learning, as evidenced by statistical tests performed with comparative results of the evaluation performed through the development of conceptual maps before and after interventions. An orientation guide was also produced to be used by professionals in the area to teach OOP.

Keywords: Computer education. Object Oriented Programming. Games development.

LISTAS DE ILUSTRAÇÕES

LISTAS DE TABELAS

LISTAS DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

A existência humana passou por várias mudanças ao longo de sua história e, nos dias atuais, o que podemos perceber é que a transformação tecnológica se tornou algo muito presente em nossas vidas. Nesse sentido, o Mestrado Profissional em Tecnologias Educacionais em Rede da UFSM (Universidade Federal de Santa Maria) procura, de maneira livre e aberta, produzir reflexões teórico/práticas sobre inovação e democratização dos processos educativos mediados por tecnologias. Assim, de acordo com a proposta do referido programa de pós-graduação, a pesquisa aqui apresentada está vinculada linha de Desenvolvimento de Tecnologias Educacionais em Rede, a qual tem o objetivo de desenvolver metodologias e ferramentas educacionais que envolvam o uso de tecnologias e que possam proporcionar a utilização de redes computacionais com efeito nos processos educativos.

A proposta desse programa de pós-graduação é bastante pertinente para o contexto contemporâneo, pois, na atualidade pouco se faz ou se produz no âmbito profissional sem o auxílio da tecnologia. Celulares, *notebooks*, *ultrabooks*, *tablets*, *smartphones*, e uma gama de outros aparelhos eletrônicos, estão presentes no convívio diário das pessoas.

Diante disso as instituições de ensino vêm se adequando a essa realidade, incorporando tecnologias digitais à sua estrutura e, na medida do possível, capacitando professores para atuar frente às enormes possibilidades do uso de tecnologias de informação e comunicação como ferramentas para potencializar a problematização dos processos de ensino e aprendizagem.

A partir dos anos 2000 a Internet chegou definitivamente na vida das pessoas, seja no convívio social, laboral ou educacional, alterando significativamente a forma de se produzir, relacionar, pesquisar e aprender. Conforme TAJRA (2009), a Internet pode ser utilizada na forma de uma biblioteca universal, composta por vasto acervo de informações do mundo todo, constantemente ampliada e aberta em tempo integral. O uso do computador na educação, aliado às possibilidades de pesquisa na Internet transformaram o cotidiano escolar, criando um movimento de ressignificação do processo educativo pela inclusão de tecnologias digitais capazes de dinamizar a pratica profissional docente.

O interesse por tais tecnologias é uma característica verificada praticamente na totalidade dos jovens em idade escolar, pois, ao mesmo em tempo que ela é útil para o desenvolvimento das tarefas do dia a dia, tem a capacidade de entreter. O entretenimento oferecido pelos meios digitais, em geral, pode ser alcançado pelo consumo de informações online, pela sociabilização oportunizada pelas redes sociais ou pelo uso de jogos. Tais situações são verificadas em diversos cursos de diversas modalidades, oferecendo aos professores inúmeras possibilidades de utilizar as tecnologias digitais em favor do melhoramento das práticas de ensino (TAJRA, 2009).

Em consonância com esse “movimento tecnológico”, desde o início, a explosão de *games* se tornou uma febre mundial. Jovens de variadas idades passam horas (que parecem minutos) diante de uma plataforma de jogo, se desafiando, aprendendo e interagindo com problemas que carecem de carga cognitiva considerável. Tais problemas possuem, em essência, potencial pedagógico significativo que nem sempre é bem aproveitado pelos usuários, considerando o estado imaturo de suas consciências intelectuais (VIGOTSKI, 2007; BECKER, 2012).

Assim, a temática para esse trabalho surge juntamente com a intenção de se utilizar o fenômeno dos jogos eletrônicos em favor da melhoria dos processos de ensino. Tal proposta é fortalecida pela sua popularidade entre os estudantes e de suas capacidades intrínsecas de motivar pessoas a resolver problemas e, de outro lado, potencializar dinâmicas de ensino nas mais diversas áreas do conhecimento (FARDO, 2013). Desse modo, o que se propõe aqui é fazer uso da motivação, atividade e desafio encontrados nos jogos (VEEN e VRAKKING, 2009), para romper as dificuldades do ensino e da aprendizagem dos conceitos de Programação Orientada a Objetos (POO).

A proposta desse trabalho foi desenvolvida no contexto do Curso Técnico em Informática Integrado ao Ensino Médio, do Instituto Federal de Educação Ciência e Tecnologia Farroupilha (IFFAR), *campus* Santo Augusto, pois a programação de computadores, englobando também a POO, é parte integrante do rol de disciplinas técnicas que são requisitos básicos aos profissionais formados pelo curso. De outro lado, muitos são os relatos de dificuldades, desistências e evasões em razão de disciplinas complexas – como é o caso da programação de computadores (SCAICO et al, 2013; SOUZA et al, 2016).

A pesquisa não se limita à utilização de jogos para facilitar a aprendizagem, o que se propõe com esse trabalho é estimular que os estudantes passem da posição

de *consumidores* para a condição de *desenvolvedores de jogos*, e que esse movimento possibilite instigá-los ao exercício da pesquisa, pois partimos do entendimento que motivá-los para isso é atributo da prática docente e missão da escola.

1.1 PROBLEMA DE PESQUISA

A escolha do tema se dá pela percepção das dificuldades que os estudantes têm em abstrair os conceitos do paradigma POO em relação ao paradigma estruturado de programação. Isso se deve pelas diferenças conceituais e estruturais característica de cada uma dessas abordagens, pois implicam em diferenças consideráveis de se abstrair o problema e principalmente na implementação sintática de sua solução. Nesse sentido, a questão norteadora desse projeto é de "**como a utilização do desenvolvimento de *jogos* pode facilitar a aprendizagem de POO no curso Técnico em Informática Integrado ao Ensino Médio do IFFAR *campus* Santo Augusto?".**

1.2 OBJETIVOS

A pesquisa tem como objetivo geral compreender como a utilização do desenvolvimento de jogos pode contribuir na apropriação dos conceitos de programação orientada a objetos. Dentre os objetivos específicos destacam-se:

a) Dinamizar os processos de ensino da disciplina de programação a fim de facilitar o processo de assimilação e apropriação dos conceitos de programação orientada a objetos;
b) Analisar as possibilidades e limites do desenvolvimento de jogos no processo de ensino e aprendizagem de POO;
c) Refletir acerca dos resultados obtidos pelo processo de ensino utilizando o desenvolvimento de jogos em favor do ensino de programação;
d) Promover a construção de itinerários pedagógicos com possibilidade de reutilização para o ensino de POO, por meio da produção de Unidades de estudo (FILATRO e CAIRO, 2015).

1.3 JUSTIFICATIVA

As transformações do ato de ensinar e de aprender sempre estiveram em voga no meio acadêmico. Todavia, esse movimento tem se intensificado a partir do advento da Internet e da popularização dos computadores em ambientes escolares. Muitos estudos estão sendo realizados sobre temas relacionados ao uso de computadores na educação, até mesmo antes da sua popularização.

O desenvolvimento de mecanismos facilitadores da aprendizagem sempre esteve em pauta de discussão. Como exemplo disso é possível citar as chamadas "*Maquinas de Ensinar de Skinner*" criadas na década de cinquenta, as quais buscavam ensinar a partir de questões cujas respostas o aluno deveria encontrar para poder avançar (MOREIRA, 2011). Embora a intenção das máquinas idealizadas por Skinner tivessem sua proposta pedagógica contestada, servem para ilustrar a histórica busca humana pela instrumentalização dos processos de ensino.

Frente à necessidade de aprimorar os métodos pedagógicos utilizados em sala de aula, os professores necessitam estar sempre à procura de meios para atender as peculiaridades existentes entre os alunos de uma classe. Desta forma, faz-se necessária a constante busca por meios capazes de oferecer uma aula atraente e que consiga alcançar os objetivos propostos, motivando os estudantes para a aprendizagem.

Ao longo dos anos muitas propostas pedagógicas inovadoras foram apresentadas, testadas e aprimoradas. Algumas permanecem vivas por muitos anos, servindo de referência, inclusive para a criação de outras. Nesse caso, é possível citar o Método Montessori[1], que na primeira metade do século XX revolucionou a forma de ensinar e aprender oferecendo aos alunos a liberdade de construir o conhecimento de acordo com o seu próprio interesse e habilidade.

Bem mais tarde, em 1993, Seymour Papert abre o prefácio do livro "*The Children's Machine: rethinking School in the Age of the Computer*" comentando que

[1] Método Montessori é o nome que se dá ao conjunto de teorias, práticas e materiais didáticos criado ou idealizado inicialmente por Maria Montessori. O método Montessori parte do concreto ao abstrato. Baseia-se na observação de que os meninos e meninas aprendem melhor pela experiência direta de procura e descoberta para tornar esse processo mais rico possível a Educadora italiana desenvolveu materiais didáticos que constitui um dos aspectos mais conhecidos de seu trabalho. De acordo com sua criadora, o ponto mais importante do método é, que a educação se desenvolva com base na evolução da criança, e não o contrário.

adentrávamos na era da informática e que o período também poderia ser denominado “a era da aprendizagem”, com vistas a escala exponencial que as informações e possibilidades de aprendizagem estavam surgindo e ao alcance de todos (PAPERT, 1992). O autor defende em sua obra, e em tantas outras, o uso de computadores na educação, sendo considerado um dos maiores visionários dessa tendência, hoje uma realidade.

Em consonância com essa linha de pensamento (VALENTE, 1993) é categórico em afirmar que “o computador é capaz de ensinar”, expressando na frase a empolgação visível nos ambientes em que os estudantes podiam interagir com tais máquinas. Papert (2008) enfatiza que as crianças do mundo inteiro passam a assumir um caso de amor com os computadores em que os utilizam para realizar as mais variadas tarefas, sendo que, a maior parte do tempo, é dedicada aos jogos.

O Paradigma Educacional Emergente, proposto por Maria Cândido Moraes (1997), também dá a base para fortalecer a intenção do uso de jogos como ferramenta de ensino e aprendizagem, pois, segundo a autora, uma das missões do docente está em preparar o indivíduo para aprender a investigar, trabalhar em grupo, dominar diferentes formas de acesso às informações, desenvolver capacidade crítica de avaliar, reunir e organizar informações mais relevantes para se chegar a melhor solução.

Do mesmo modo, Vigotski (2007) ressalta a importância da interação entre o indivíduo e o ambiente, pois, considera fundamental utilizar ferramentas que permitam visualizar na prática as causas e efeitos de suas produções. Nesse caso, o autor destaca, que a relação de um indivíduo com o mundo se dá através de instrumentos e da linguagem e, que a aprendizagem se dá com a presença de um mediador, que tende a problematizar o processo, mas quem conduz a própria aprendizagem é o aluno.

Por se tratar da aprendizagem de linguagens de programação para computadores, nada é mais adequado do que aliar os processos de ensino e de aprendizagem ao uso de ferramentas computacionais, pois essas constituem o cerne que norteia a ciência estudada pelos envolvidos. Para além disso, a utilização adequada e criativa de tais ferramentas possibilita também, ensinar aprendendo, a partir da construção coletiva dos saberes e da estimulação das potencialidades dos discentes.

Ao pesquisar sobre o ensino de programação em eventos e periódicos especializados em educação e informática, muitas experiências sobre métodos e dinâmicas são relatadas, porém, o que se verifica nas abordagens utilizadas, em sua grande maioria, é que estão voltadas ao ensino e aprendizagem de conceitos introdutórios de programação ou do paradigma de programação estruturada, pouco se explorando o paradigma de Orientação a Objetos (OO), que é alvo deste trabalho.

A programação estruturada é baseada no desenvolvimento de programas utilizando apenas estruturas sequenciais, condicionais e de repetição. Já a POO além das estruturas do primeiro, agregam em sua sintaxe conceitos de classe, objetos, herança, encapsulamento, polimorfismo, etc. Nesse caso, o ensino de programação orientada a objetos (POO) apresenta dificuldades ainda maiores, pois estudantes necessitam abstrair problemas computacionais a partir da mudança de paradigma que envolve os conceitos de OO, considerando que, usualmente, iniciam sua exploração em algoritmos e programação pelo paradigma estruturado.

Tais problemas são também constatados no curso Técnico em Informática do IFFAR *Campus* Santo Augusto, pois estudantes apresentam grandes dificuldades em compreender a mudança estrutural na forma de programar. Nesse sentido a realização dessa pesquisa, tem também a intenção de oferecer ao pesquisador e pesquisados melhorias nos processos de ensinar e aprender.

1.4 ESTRUTURA DO TRABALHO

Para conduzir essa Dissertação, o texto que segue está organizado em seis capítulos.

No Capítulo 2 é apresentado um referencial teórico acerca do paradigma de programação orientado a objetos, abordando os principais conceitos e a sua importância para o mundo do desenvolvimento de *software*. Também são abordados aspectos relacionados à importância de estudar programação de computadores, a fim de obter crescimento pessoal e habilitar-se para as oportunidades profissionais inerentes a essa formação. Ao final do Capitulo 2 é realizada uma breve contextualização acerca do perfil do estudante da atualidade, do uso das TIC na educação e a influência dos jogos na aprendizagem.

No Capítulo 3 é conduzida uma revisão sistemática da literatura em busca de conhecer as principais ferramentas e estratégias de ensino de programação a nível nacional.

O Capítulo 4 trata do referencial metodológico utilizado na presente pesquisa, discriminando os materiais e métodos utilizados.

O Capítulo 5 traz os resultados obtidos de todo o estudo, discriminando, detalhadamente as intervenções realizadas, tabulação e análise quanti-qualitativa do processo de avaliação do produto proposto na utilização de Unidades de estudo, Greenfoot, Ciclo de aprendizagem de Kolb e Mapas Conceituais para o ensino e aprendizagem de POO.

Por fim, o Capitulo 6 apresenta as considerações finais desse estudo.

2 REFERENCIAL TEÓRICO

Este capítulo aborda os conceitos necessários para que se entenda os princípios norteadores desse trabalho. Serão apresentadas as concepções históricas e elementos fundamentais relacionados à programação de computadores e POO. Ainda, é realizada uma breve contextualização acerca das motivações para aprender sobre linguagens de programação e o uso de tecnologias de informação e comunicação e jogos para auxiliar na aprendizagem.

2.1 PROGRAMAÇÃO DE COMPUTADORES: UM BREVE HISTÓRICO

Para entender a POO é necessário resgatar uma pequena parcela do processo histórico relacionado à evolução das linguagens de programação. Nesse caso, necessitamos olhar para o passado e entender como o computador foi concebido.

A história começa por volta de 1821, quando Charles Babbage constrói a **Máquina Diferencial**. O termo “computador” nem existia na época, mas a referida máquina era capaz de solucionar equações trigonométricas, polinomiais e logarítmicas. O diferencial dessa máquina em relação a outras existentes era a característica inovadora de permitir diferentes configurações. Tais configurações permitiam processamentos diferenciados através de um “programa” (ANSELMO, 2013). A criação da Máquina Analítica dá a Babbage o título, reconhecido por muitos, de “pai do computador”.

Mais tarde, Augusta Ada Byron[2] (Ada Lovelace) – formada em Matemática e Metafísica – desenvolve pesquisas relacionadas ao desenvolvimento de máquinas de calcular, atuando em conjunto com Babbage na construção da Máquina Analítica.

> Ada é a primeira pessoa no mundo a mencionar IA (inteligência Artificial) e dedicou sua vida ao projeto de máquinas para a realização de cálculos matemáticos. Ada trocou diversas cartas com Charles Babbage durante o desenvolvimento de sua Máquina Analítica e escreveu notas extensas de seu projeto juntamente com vários programas matemáticos complexos que

[2] Augusta Ada Byron - Ada Lovelace (1815 a 1852): foi uma mulher à frente do seu tempo dedicando sua vida ao estudo de assuntos que na época eram exclusivos aos homens. Sua notória influência para o estudo da computação foi reconhecida no ano de 1980 pelo Departamento de Defesa dos EUA pela criação de uma linguagem de programação ao qual batizaram com o nome “Ada” em sua homenagem. (ANSELMO, 2013).

> levaram muitas pessoas a caracterizá-la como a primeira programadora (ANSELMO, 2013, p.20, grifo no original).

A exemplo do título moral de Babbage, Ada é mundialmente reconhecida pela sua contribuição, dedicação e perseverança ao insistir em estudar assuntos de uma área em que, no seu tempo, era dominada pelo gênero masculino e a inclusão de mulheres nesse cenário era impensável. Além de ser considerada a primeira programadora, também é considerada por muitos como a "mãe da lógica computacional" por criar conceitos fundamentais da programação de computadores, tais como *salto condicional* e *estruturas de repetição*.

Depois disso, muitas pesquisas surgiram, tiveram sucesso, outras nem tanto. Mas foi no período da Segunda Guerra Mundial (1939 a 1945) que muito se produziu no ramo de tecnologia. É fato que boa parte de tais produções eram fomentadas por interesses bélicos e, nesse caso, a evolução de ambientes computacionais a serem utilizados na automação de processos de construção de armamentos ocorreu de maneira ampla. Destaques ficam por conta do aprimoramento das telecomunicações, radio difusão e, em especial o desenvolvimento dos primeiros computadores eletrônicos. Nesse caso, pode-se citar o surgimento do ENIAC (*Eletronic Numerical Interpreter and calculator*), concebido logo após o término da Segunda Guerra Mundial com a intenção de realizar complexos cálculos balísticos de maneira automática (MONTEIRO, 2007).

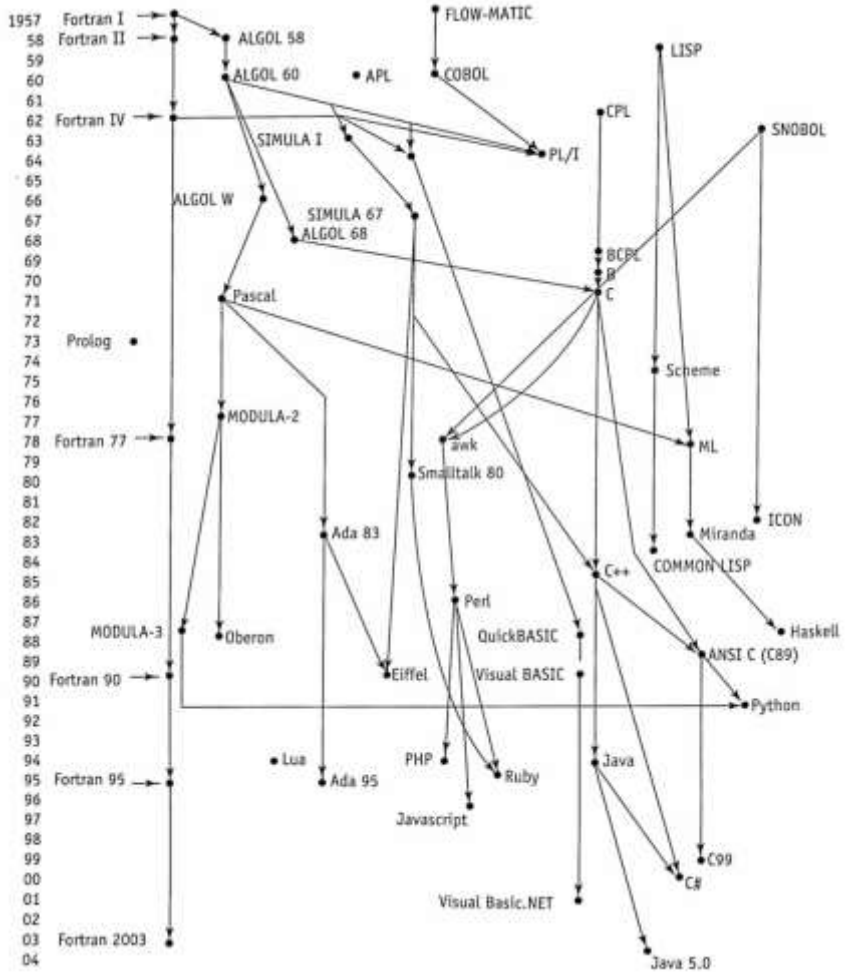
Nessa época, embora o computador estivesse em voga no campo científico, ainda não havia linguagens de programação com conhecemos na atualidade, pois as linguagens eram de baixo nível e eram pensadas para atender necessidades de computadores específicos. O exemplo disso, temos a linguagem *Plankakül de Zuse* que fora desenvolvida em 1945 pelo cientista alemão Konrad Zuse para ser utilizada no computador eletromecânico que ele mesmo desenvolveu e batizou de Z4. A linguagem *Plankakül* nunca foi implementada e sua documentação só foi publicada no ano de 1972, mas sua estrutura e estudos serviram de base para o desenvolvimento de outras frente à riqueza de detalhes em vários aspectos (SEBESTA, 2011).

No ano de 1954 a evolução e miniaturização do *hardware* possibilitaram que a IBM construísse o 704 e, juntamente com ele, nasceu a linguagem de programação FORTRAN, a qual foi apresentada por meio do relatório intitulado: "The IBM Mathematical FORmula TRANslating System: FORTRAN" (MONTEIRO, 2007;

SEBESTA, 2011). Essa linguagem de programação teve o diferencial das demais existentes por ser de alto nível e compilada, enquanto que as demais linguagens eram de baixo nível e, em sua grande maioria, interpretadas.

Assim, novas linguagens de programação foram surgindo e sendo aprimoradas com o passar dos anos, cada uma visando atender necessidades específicas, a evolução do hardware ou paradigmáticas da sua época. A Figura 1 ilustra o surgimento, evolução e influência das principais linguagens conhecidas.

Figura 1 - Genealogia das principais linguagens de programação de alto nível



Fonte: Sebesta (2011).

Depois da criação do ENIAC e o desenvolvimento do transistor, computadores puderam ser construídos de maneira mais compacta, assim, inicia-se um movimento a nível mundial pelo desenvolvimento de sistemas e programas capazes de automatizar processos da vida contemporânea. O período dos anos 1960 a 1980 é

marcado por um frenesi pela criação de novas linguagens de programação e aprimoramento das existentes, pois, é nesse período que surgem as linguagens: C, Fortran (várias versões), ALGOL, PASCAL, COBOL, etc (MONTEIRO, 2007; SEBESTA, 2011; DEITEL e DEITEL, 2017).

Em geral, cada período da história recente da computação é marcado pela mudança frequente de paradigmas. Anselmo (2013) aborda essa constatação elencando tendências verificadas ao longo desse período, em intervalos de cinco anos. Segundo o autor, os anos 1980 são marcados pelo desenvolvimento de *software* com destaque para linguagens como Fortran, Algol, Cobol, Pascal e C – sendo o programador o elemento fundamental dessa tendência. Já nos anos de 1985 os dados ganham atenção especial e a implementação da linguagem SQL (*Structure Query Language*) e ferramentas CASE (*Computer-Aided Software Engineering*) possibilitaram a automatização de processos, marcando esse período.

Em 1990 a "produtividade" é a palavra de ordem e o desenvolvimento acelerado está em voga. Modelo *cliente/server*, RAD (*Rapid Application Development*), juntamente com as ferramentas de desenvolvimento do tipo "arrastar e soltar" são as mais procuradas, nesse caso *Visual Basic* e Delphi ganham destaque. No ano de 1995 a *Internet* ganha o *status* de "divindade" na área de computação. Com ela ocorrem os surgimentos dos conceitos de "Computação Distribuída" e "Escalabilidade". Nesse período, também, o paradigma de OO é firmemente divulgado como melhor modelo de desenvolvimento (ANSELMO, 2013).

Os anos 2000 por sua vez, frente a inquietações causadas pelo "*Bug* do Milênio" tornam o "Controle" o foco das atenções. Assim, todo o processo de desenvolvimento de *software* deve passar rigorosamente por fases da Engenharia de Software. Em 2005 a *Web* ganha novamente destaque com o surgimento das Redes Sociais e mensageiros instantâneos - assim, linguagens como ASP, JSP e PHP recebem especial atenção. Em 2010 os dispositivos móveis estão no centro das atenções com a disseminação de bilhões de equipamentos portáteis pelo mundo todo e, com a maioria conectada à internet o desenvolvimento para esses dispositivos está em destaque (ANSELMO, 2013).

Ao concluir essa seção do livro, Anselmo (2013) deixa uma previsão sobre a tendência para 2015 – acertando em grande parte – pois o autor sugere que 2015 seria o ano da "interatividade", do "*cloud computing*", do "*big data*" da mineração de dados de navegação de usuários possibilitando traçar seus perfis de compras e, dessa

maneira atuar de forma a oferecer ao mercado poderoso sistema de informação acerca das tendências da clientela. Com essas previsões, o autor não erra em nada, todavia, outros elementos poderiam entrar na lista, pois, esse período é marcado também pelo aumento das vendas online, o compartilhamento de conteúdo, a internet das coisas (IoT – *internet of things*), e tantas outros.

2.2 O PARADIGMA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

No desenvolvimento desse capítulo realizou-se uma breve contextualização histórica acerca do computador, das linguagens de programação e das rápidas transformações características do mundo da computação. Todavia, faz-se necessário abordar mais especificamente o tema que é objeto de estudo deste trabalho - a Programação Orientada a Objetos. Assim, para dar prosseguimento a essa pesquisa, necessita-se compreender o que é um “paradigma de programação”.

Nesse caso, em linguagem de programação, tem-se que um paradigma é um ponto de vista pelo qual um problema computacional é abordado. Tal abordagem é o que determinará a construção sintática de uma solução. Na atualidade existem vários paradigmas sugeridos por diversas bibliografias, porém Tucker e Noonan (2010) elencam quatro principais: *Imperativo, Orientado a Objetos, Funcional e Lógico*, sendo que cada um determina uma forma particular de abordar os problemas e de formular soluções. De outro lado, Sebesta (2011) destaca também que uma linguagem de programação pode suportar mais de um paradigma para ampliar possibilidades de soluções.

No caso do paradigma de *Programação Orientada a Objetos,* este tem sua origem a partir do final dos anos 1960 pela conclusão da tese de Doutorado de Alan Kay[3], pela Universidade de Utah, ao desenvolver pesquisas com a linguagem SIMULA[4]. Os estudos de Alan previam a disponibilidade de computadores de mesa

[3] Alan Kay - é um especialista em tecnologia da informação estadunidense reconhecido por ter sido um dos inventores da linguagem de programação Smalltalk, e um dos pais do conceito de programação orientada a objetos.
[4] SIMULA é uma linguagem de programação, projetada para apoiar a simulação de eventos discretos, criada entre 1962 baseada em ALGOL 60 serviu de base para Alan Kay no desenvolvimento de programação orientada a objetos.

poderosos capaz de processar centenas de milhares de instruções por segundo e teriam disponível uma memória RAM com muitos megabytes para armazenar instruções de programas utilizados pelo usuário. A primeira linguagem a oferecer suporte completo a POO foi Smalltalk no ano de 1980 (SEBESTA, 2011). Na atualidade, a POO é um dos paradigmas mais importantes no desenvolvimento de software, pois linguagens amplamente utilizadas suportam essa abordagem, dentre elas pode-se citar Perl, Python, PHP, Java, C++ e C#. A programação orientada a objetos pode ser definida, segundo Santos (2003), da seguinte maneira:

> Programação Orientada a Objetos ou, abreviadamente, POO, é um paradigma de programação de computadores onde se usam classes e objetos, criados [...] para representar e processar dados usando programas de computadores.

A proposta desse paradigma está na naturalização da programação de computadores, de forma que entidades da vida real possam ser efetivamente projetadas e implementadas em ambiente computacional. Para além do desenvolvimento de software, os conceitos de OO são extremamente úteis na modelagem de sistemas e bases de dados, assim, trata-se de um paradigma de análise, projeto e programação de sistemas de informação (SANTOS, 2003).

A naturalização no desenvolvimento de software foi um movimento iniciado na época da sua idealização, com vistas na sua humanização de processos computacionais visto que esta área estava restrita ao campo científico e a programação de computadores era caracterizada como uma atividade difícil e extremamente técnica. A tentativa de humanizar o desenvolvimento de software é iniciativa de alguns visionários da época, como o próprio Alan Kay, o qual recebe influências de Seymour Papert, cujas pesquisas estão voltadas no desenvolvimento de possibilidades de uso do computador em favor da aprendizagem com a concepção da linguagem LOGO (SEBESTA, 2011).

Em relação à POO, Joyanes Aguilar (2008) destaca três pontos fundamentais:

a) A POO utiliza objetos, não algoritmos, como bloco de construção lógica;
b) Cada objeto é uma instância de uma classe;
c) As classes se relacionam umas com as outras por meio de uma relação de herança.

De maneira geral, a POO oferece muitas vantagens em relação a outras abordagens, pois, a alta reutilização de código permite significativa redução tempo de produção e manutenção de *software* e, a redução da sua complexidade pela

abstração do sistema. Todas essas vantagens culminam pelo aumento qualidade e produtividade dos produtos desenvolvidos. Assim, para melhor caracterizar essa abordagem, Joyanes Aguilar (2008) elenca cinco conceitos fundamentais da POO: *objetos, classes, herança, mensagem, polimorfismo e encapsulamento*, que serão apresentados nas próximas subseções.

2.2.1 Objetos, atributos, métodos e instanciação

No mundo real existem muitos elementos que interagem entre si, sendo que cada um desempenha funções específicas em relação ao seu contexto. Tais elementos são objetos. Assim, objetos são “coisas” e podem ser concretos ou abstratos. Um livro, um notebook, uma cadeira ou um veículo são exemplos de objetos concretos comuns do cotidiano das pessoas. De outro lado, uma conta em um banco ou uma equação matemática são exemplos de objetos abstratos que também podem ser implementados em ambiente computacional (CARDOSO, 2006; KEOGH e GIANNINI, 2005; DEITEL e DEITEL, 2017).

Assim, se chega ao ponto fundamental desse paradigma de desenvolvimento, pois ele considera a transformação de objetos reais em objetos computacionais. Desse modo, tais elementos são estruturas de programas que contém as características e os comportamentos que representam um dado objeto dentro do sistema. A ideia central de POO gira em torno dos objetos, sendo eles os elementos essenciais do desenvolvimento sob essa perspectiva. Cada objeto combina em cada elemento singular várias informações (atributos) e operações (métodos).

Em programação orientada a objetos, tem-se que **atributos** são as características particulares de cada objeto. Tais características são definidas na classe a que o objeto pertence e, dessa forma, todos os objetos dessa classe compartilham das mesmas características, porém, com valores diferentes (CARDOSO, 2005; DEITEL e DEITEL, 2017). Juntamente com os atributos, cada objeto pode realizar operações, as quais são chamadas **métodos**.

> O método armazena as declarações do programa que, na verdade, executam as tarefas; além disso, ele oculta essas declarações do usuário, assim como o pedal do acelerador de um carro oculta do motorista os mecanismos para fazer o veículo ir mais rápido (DEITEL e DEITEL, 2017).

Em síntese, um objeto é uma entidade capaz de reter um estado (através de seus atributos) e que oferece uma série de métodos capazes de examinar ou afetar

este estado. Dentro do contexto de execução de um *software*, para organizá-los, no momento da sua criação lhe é definido um nome ou identificador, exemplo: “carro1” (JOYANES AGUILAR, 2008).

É importante observar que a criação de objetos durante a execução do programa requer um comando de criação, que segue a mesma sintaxe da utilização de variáveis. Todavia, vale ressaltar que a criação de um objeto não se trata apenas de uma declaração de variável, pois o processo necessita de comando específico o qual se denomina **instanciação** (SANTOS, 2003).

2.2.2 Classe

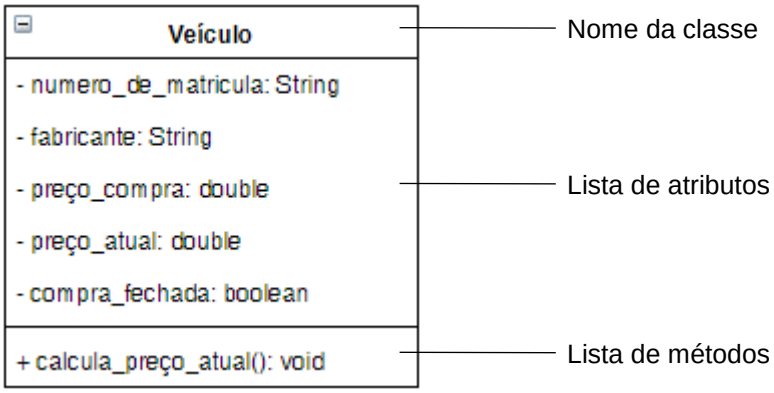
Classes são modelos pelos quais os objetos são descritos. É na classe que o desenvolvedor define todas as características e operações que farão parte dos objetos que dela tiverem origem. Assim, uma classe é uma descrição de um conjunto de objetos, pois nela constam as especificações de atributos e que reúnem características comuns deste conjunto (SANTOS, 2003).

Na classe, o desenvolvedor determina quais serão os atributos e ações de cada objeto dentro do contexto computacional, além disso, é dentro da descrição da classe que o desenvolvedor estabelece, por meio de padrões de acesso e visibilidade (público ou privado), as regras de encapsulamento das informações e métodos.

Previamente à implementação de uma classe, é realizada a sua modelagem conforme regras estabelecidas pelos padrões de desenvolvimento de software, que atualmente é realizada pela UML, a qual tem suas especificações padronizadas pela OMG[5]. A Figura 2 ilustra a representação UML de uma classe hipotética “Veículo”.

[5] O Object Management Group® (OMG®) é uma associação aberta internacional, sem fins lucrativos, que tem o objetivo de desenvolver padrões internacionais de tecnologia para usuários do mundo inteiro. Essa associação foi fundada em 1989 e, desde então, fornece padrões OMG direcionados à fornecedores, usuários finais, instituições acadêmicas e agências governamentais (OMG, 2017).

Figura 2 – Representação UML de uma classe



Fonte: do autor

Assim, como se pode verificar na Figura 4, cada objeto criado a partir dessa classe, terá os mesmos atributos e métodos nela descritos. O que será diferente de um objeto em relação a outro será o seu identificador e o seu “estado” que é formado pelos valores armazenados em seus atributos.

A abstração é uma característica fundamental da programação e do projeto orientados a objetos, pois é por meio dela que o projetista tem a ideia geral da proposta e dos objetivos do produto a ser desenvolvido. Essa característica permite que grandes sistemas sejam especificados em um nível global, antes de ocorrer a implementação dos métodos individuais (TUCKER e NOONAM, 2010).

2.2.3 Herança

A característica de herança pode ser considerada um dos principais conceitos do paradigma orientado a objetos (SANTOS, 2003), pois, por intermédio dela é possível a reutilização de código anteriormente desenvolvido. Essa característica agiliza o processo de desenvolvimento e manutenção de produtos em linguagem orientada a objetos. Todavia, é importante ressaltar que a herança não se restringe à possibilidade de reutilização de códigos, pois também, proporciona ao projetista e ao desenvolvedor maior clareza e organização durante todas as fases do processo de criação do software.

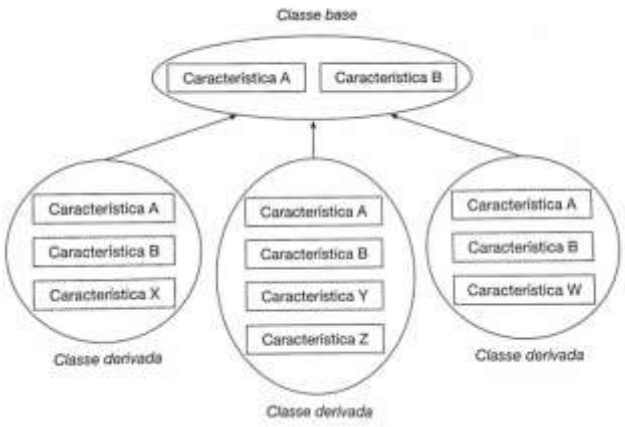
Em relação à herança, Tucker e Noonam (2010, p. 319) destacam:

> O paradigma orientado a objetos suporta reutilização de código por intermédio da *herança*. As classes existem em uma linguagem orientada a objetos em uma hierarquia de classes. Uma classe pode ser declarada como uma *subclasse* de outra classe, que é chamada de *classe-mãe* ou *superclasse*. Dentro dessa hierarquia, cada subclasse é capaz de herdar

> variáveis e métodos de sua classe-mãe simplesmente em virtude do fato de que ela é uma subclasse (grifo no original).

Segundo Joyanes Aguilar (2008, p.554), a herança se apoia no significado desse conceito na vida diária, de modo que as classes mais básicas se dividem em subclasses assim como os animais se dividem em: mamíferos, anfíbios, insetos, peixes, etc. Porém, o que é importante no princípio de herança, como na biologia, é que as classes filhas carregam junto consigo as características – no caso computacional: atributos e métodos – da classe-mãe (veja Figura 3).

Figura 3 - Hierarquia de classes



Fonte: Joyanes Aguilar (2008).

Assim, a herança supõe a existência de uma classe principal (classe base) e uma hierarquia abaixo dela formada por classes derivadas. As classes derivadas (subclasses) herdam o código da superclasse acrescentando-lhe características próprias na forma de características (atributos), de operações (métodos) ou pelo aperfeiçoamento/adequação dessas operações.

2.2.4 Mensagem

Mensagens são a forma de comunicação entre objetos, por meio delas é possível acessar informações retidas no estado de um objeto. As mensagens são responsáveis por ativar os métodos que residem nos objetos, pois são eles – os métodos – que definem como um determinado objeto deve reagir às mensagens a ele enviadas, representando o seu comportamento (DEITEL e DEITEL, 2017).

A estrutura interna dos métodos está oculta aos usuários (encapsulada) sendo as mensagens os únicos condutores que conectam os objetos ao meio externo. Assim mensagens e métodos podem ser caracterizadas como “caras da mesma moeda”,

pois estão intimamente relacionados, um complementando a existência do outro (JOYANES AGUILAR, 2008).

A estrutura de uma mensagem, em geral, é composta de três partes:

a) Destino: o objeto receptor;
b) Método: a operação que deve ser realizada;
c) Parâmetros: as informações necessárias à execução do método.

De maneira geral o protocolo de mensagens envolve duas partes essências: *emissor* e *receptor*. Assim, o objeto que envia uma mensagem é chamado de "emissor" e o objeto de destino é o "receptor".

2.2.5 Polimorfismo

Polimorfismo significa "várias formas" e é uma característica fundamental da POO, pois é a capacidade de modificar comportamentos de métodos dos objetos. Segundo (Tucker e Noonam, 2010, p. 323) "em linguagens orientadas a objetos, polimorfismo refere-se à ligação tardia de uma chamada a uma ou várias diferentes implementações de um método em uma hierarquia de herança". Assim, o conceito de polimorfismo está relacionado com a possibilidade da relação de herança entre as classes, pois, a mudança de comportamentos de objetos pertencentes a uma mesma estrutura hierárquica e, dispostos em diferentes níveis, é o que possibilita o polimorfismo.

Essa característica da POO pode trazer muitos benefícios ao desenvolvimento, podendo ser encarada sob a perspectiva de compatibilidade de objetos. Um exemplo disso está na situação hipotética de um método receber por parâmetro um objeto do tipo "Veículo" e, considerando que "Carro", "Caminhão" ou "Motocicleta" são descendentes de "Veículo". Desse modo, qualquer um desses objetos poderá ser passado como parâmetro do referido método, visto que possuem a mesma estrutura (SANTOS, 2003).

De maneira geral, o polimorfismo pode acontecer por meio da sobrecarga[6] de métodos ou da sua reescrita[7]. Em uma situação mais extrema, considerando que um

[6] A Sobrecarga (*overload*) usa o número ou tipo de argumento para distinguir entre funções ou operadores com nomes idênticos. Pode-se ainda dizer que é a capacidade de poder definir métodos ou procedimentos em uma classe com o mesmo nome, mas parâmetros diferentes.
[7] Possibilidade de alterar comportamento de métodos em classes descendentes pela sua reescrita (reescrever, sobrescrever, *override*).

objeto é um tipo de dado, objetos já instanciados podem sofrer mutação de tipo através de *casting*, desde que exista uma relação de herança entre a classe em que o objeto foi criado e a classe que se pretende transformá-lo.

2.2.6 Encapsulamento

O encapsulamento é a técnica utilizada para restringir o acesso aos atributos, métodos ou até mesmo à própria classe. Nesse caso, os detalhes da implementação ficam ocultos ao usuário da classe, dessa forma, o usuário passa a utilizar os métodos de uma classe sem se preocupar com detalhes sobre como o método foi implementado internamente (CARVALHO e TEIXEIRA, 2012).

Ocultar aspectos que não precisam ser mostrados ao usuário é caracterizado como um grande trunfo da POO em relação a outros paradigmas de programação, pois dessa maneira, a complexidade do código permanece transparente para o usuário, de forma que apenas reutiliza a interface previamente implementada.

O encapsulamento de atributos, métodos ou classes se dá por meio da utilização de padrões de acesso, os quais são definidos no momento em que tais elementos são implementados. Em POO podem ser utilizados três formatos de padrão de acesso: público; privado e protegido. Tais elementos podem ser descritos da seguinte maneira:

> *public* (público): indica que o método ou o atributo são acessíveis por qualquer classe, ou seja, que podem ser usados por qualquer classe, independentemente de estarem no mesmo pacote ou estarem na mesma hierarquia;
>
> *private* (privado): indica que o método ou o atributo são acessíveis apenas pela própria classe, ou seja, só podem ser utilizados por métodos da própria classe;
>
> *protected* (protegido): indica que o atributo ou o método são acessíveis pela própria classe, por classes do mesmo pacote ou classes da mesma hierarquia [...] (CARVALHO e TEIXEIRA, 2012, grifo no original).

Contudo, o acesso e utilização dos atributos encapsulados de uma classe é possível por meio dos métodos acessores. Tais métodos têm a função de servir de interface de acesso ao conteúdo existente nos atributos da classe. Nesse caso, atributos encapsulados devem ter um método que obtenha o seu valor atual (método *get*) e um método que possibilite alterar o valor do atributo (método *set*) (CARVALHO e TEIXEIRA, 2012).

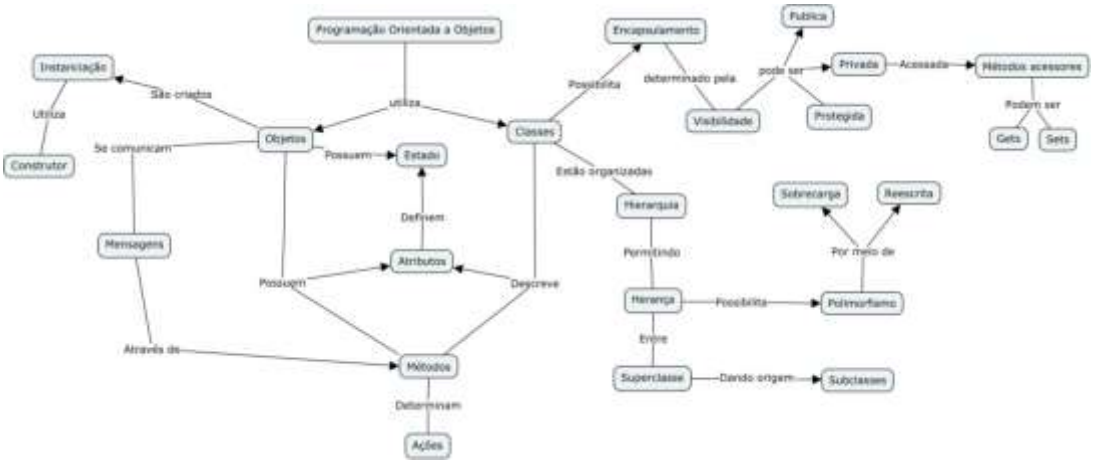
2.2.6 Considerações da seção

Em síntese, pode-se dizer que a POO utiliza classes e objetos para o desenvolvimento de software, sendo que cada objeto possui atributos e métodos encapsulados em sua estrutura. Os atributos – que são as características dos objetos – definem o seu estado. Já os métodos determinam as ações que esse objeto pode realizar dentro de um sistema. Tais ações são invocadas pelo envio de mensagens entre os objetos. Objetos são criados por meio de instanciação que utiliza o construtor para definir o estado inicial dos objetos.

As classes são responsáveis por descrever os atributos e métodos que farão parte de cada objeto criado a partir dessa classe. Estão organizadas em uma estrutura hierárquica que possibilita estabelecer uma relação de herança entre super e subclasses. A herança, por sua vez, possibilita a condição de objetos se tornarem polimorfos por meio de sobrecarga e reescrita de seus métodos. É durante a descrição da classe que o desenvolvedor define, por meio de padrões visibilidade, quais elementos dos objetos – atributos e métodos – serão públicos ou privados, assim, é definido o encapsulamento da classe.

A Figura 4 procura descrever em um mapa conceitual essa síntese, elencando os principais conceitos relacionados à POO de forma a criação de premissas conceituais acerca do tema.

Figura 4 - Mapa conceitual POO



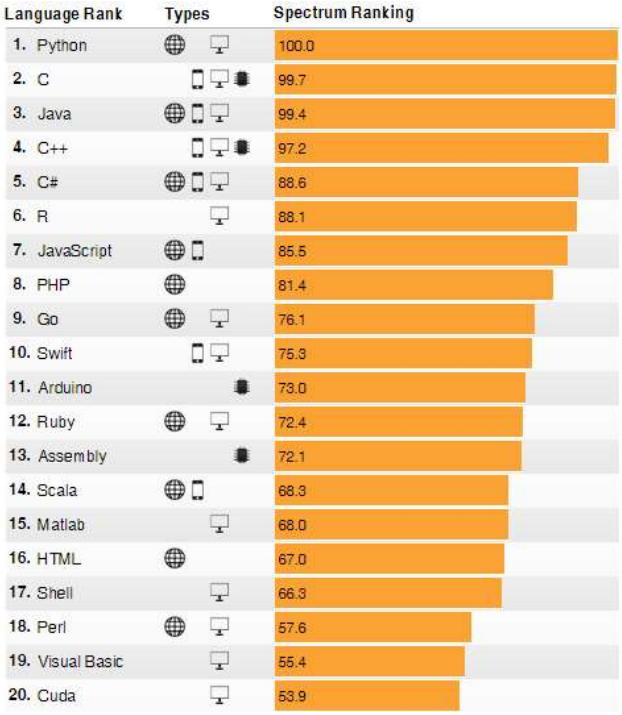
Fonte: do autor.

Diante de todas essas características a POO tem sido apontada por muitos autores como a melhor abordagem para o desenvolvimento de sistemas (ANSELMO,

2013) e, nesse caso, espera-se que o profissional da área da computação compreenda de maneira adequada os conceitos relacionados à sua área de formação, no caso da POO, trata-se de uma abordagem fundamental a qualquer programador da atualidade.

Recentemente, a IEEE[8] publicou uma pesquisa sobre as linguagens de programação mais utilizadas – lembrando que nem sempre isso significa que são as melhores – elencando um *ranking* a nível mundial, conforme ilustra a Figura 5.

Figura 5 - Principais linguagens de programação 2017



Fonte: IEEE(2017a) com adaptações.

O *ranking* apresentado pelo portal do IEEE fornece informações úteis para desenvolvedores e especialistas na área. Nesse caso, pode-se perceber que dentre as dez linguagens mais utilizadas sete suportam POO. Essa constatação aponta para a necessidade de conhecer a fundo os conceitos desse paradigma de desenvolvimento, pois está presente nas mais importantes linguagens de programação da atualidade.

[8] O *Institute of Electrical and Electronics Engineers* ou Instituto de Engenheiros Eletricistas e Eletrônicos (português brasileiro) é uma organização profissional sem fins lucrativos, fundada nos Estados Unidos. A IEEE é a maior organização profissional técnica do mundo dedicada ao avanço da tecnologia em benefício da humanidade (IEEE, 2017b).

2.3 POR QUE ESTUDAR LINGUAGENS DE PROGRAMAÇÃO?

Frente a esse conjunto de fatos e tendências é importante que se entenda a importância do estudo de programação de computadores, pois, embora muitos estudantes da área a considerem uma tarefa difícil, nela reside um vasto nicho profissional a ser explorado por pessoas interessadas a entrar para o ramo da computação.

Segundo estudo realizado por Brackmann (2017, p.42) “há uma alta demanda de mão de obra qualificada com conhecimentos em programação”. Essa é uma das preocupações de grandes multinacionais da área de tecnologia (Microsoft, Google e Facebook) as quais manifestaram essa constatação por meio de uma carta aberta assinada pelos funcionários do alto escalão.

Da mesma forma, compreender os conceitos de linguagens de programação é pré-requisito imprescindível aos profissionais da área, pois a grande maioria dos produtos computacionais gira em torno do desenvolvimento de *software*. Ademais, compreender sobre diversos assuntos é qualidade de um profissional completo e apto a competir com propriedade no mundo do trabalho.

Nessa linha de pensamento Sebesta (2011) destaca seis razões para que estudantes da área de computação dediquem-se no estudo de conceitos de linguagens de programação:

a) *Capacidade aumentada para expressar ideias*;
b) *Embasamento para escolher linguagens adequadas*;
c) *Habilidade aumentada para aprender novas linguagens;*
d) *Melhor entendimento da importância da implementação*;
e) *Melhor uso de linguagens já conhecidas*;
f) *Avanço geral da computação.*

De outro lado, pelo viés da formação integral do ser humano, compreender sobre linguagens de programação traz, segundo Papert (1971), melhorias nos processos de assimilação e do raciocínio lógico, pois, nesse caso extrapola-se a ideia de solucionar problemas puramente pelo viés “teoremático” – comumente utilizado no cotidiano escolar – sendo que soluções computacionais requerem, em geral, a construção a partir da problematização ou o exercício dela.

Sobre essa temática, Clareto (2013, p.sn) descreve:

> O teorema e o problema ou o teoremático e o problemático. O teoremático: definições ou noções primitivas, axiomas, teoremas; verdades a serem demonstradas; caminhos a serem seguidos. Matemática régia. Matemática maior. O problemático: não mais obstáculo, mas ultrapassagem, travessia; afecção, acontecimento. Enquanto o teoremático opera com definições e deduções, o problemático experimenta, experiencia.

Nesse caso, há uma diferença significativa na implementação de soluções, pois, a partir da abordagem teoremática, a solução é estanque – definida a partir da manipulação de valores variáveis em um arcabouço. Enquanto que de outro lado, o viés problemático "invade" diferentes áreas do conhecimento a fim de contextualizar e provocar desconforto/inquietações que necessitam de abstração conceitual entre problema e solução.

Assim, a abordagem problemática permite o aprimoramento cognitivo, pois é caracterizado como a "resistência aos processos instituídos de pensar, aos modos hegemônicos de matematizar, às verdades estabelecidas pela matemática régia" (CLARETO, 2013).

Sobre essa seara, Wing (2008, p.3717) ressalta que "a essência do pensamento computacional é abstração"[9] (tradução nossa) pois opera constantemente com problemas do cotidiano que necessitam de interpretação para que a solução seja eficiente. De todo modo é importante reconhecer que o pensamento matemático puro está intrínseco no processo, porém, atua como coautor na solução do problema, pois o sucesso da implementação depende da criatividade do seu desenvolvedor em reconhecer com qualidade a problemática envolvida e possíveis falhas de execução.

> De duas maneiras, nossas abstrações tendem a ser mais ricas e mais complexas do que as ciências matemáticas e físicas. Em primeiro lugar, nossas abstrações não necessariamente apreciam as propriedades algébricas limpas, elegantes ou facilmente definíveis das abstrações matemáticas, como números reais ou conjuntos, do mundo físico. [...] Segundo, porque nossas abstrações são finalmente implementadas para funcionar dentro das restrições do mundo físico, temos que nos preocupar com a finalidade e casos de falha (WING, 2008, p.3717-3718, tradução nossa).[10]

As colocações da autora corroboram com a obra de Seymour Papert (1985) ao incentivar o uso do computador para promover condições de aprendizagem,

---

[9] "The essence of computational thinking is abstraction."

[10] "In two ways, our abstractions tend to be richer and more complex than those in the mathematical and physical sciences. First, our abstractions do not necessarily enjoy the clean, elegant or easily definable algebraic properties of mathematical abstractions, such as real numbers or sets, of the physical world.[…] Second, because our abstractions are ultimately implemented to work within the constraints of the physical world, we have to worry about edge cases and failure cases."

sobretudo em estimular uma nova forma de aprender pelo exercício da abstração de problemas cotidianos. Nesse contexto, o autor comenta, na apresentação do livro *Logo: computadores e educação*, a sua afinidade com as engrenagens do sistema diferencial em que costumava brincar enquanto criança – nesse texto as engrenagens ganham destaque por proporcionar, diferentes resultados a partir de diferentes combinações e o fazem constatar que “um sistema podia ser legítimo e completamente compreensível sem ser rigidamente determinístico” (PAPERT, 1985, p.12). Nesse aspecto é que Jeannette Wing e Seymour Papert coincidem suas linhas de pensamento, por reconhecer que problemas do dia a dia são orgânicos, pertencem a um complexo e, para solucioná-los necessitamos compreendê-los em sua forma abstrata, na qual a matemática pura não consegue alcançar.

Seymour Papert, pesquisador-referência quanto ao uso de uma linguagem de programação para estimular o raciocínio lógico, deixa claro que a intencionalidade de se incentivar que crianças saibam programar uma máquina vai além de instigar que estas pensem “como máquinas” e sim que desenvolvam habilidades sobre “como pensar” um problema:

> Embora a tecnologia desempenhe um papel essencial na realização de minha visão sobre o futuro da educação, meu foco central não e a máquina mas a mente e, particularmente a forma em que movimentos intelectuais e culturais se auto definem e crescem. Na verdade, o papel que atribuo ao computador é o de um portador de "germes" ou "sementes" culturais cujos produtos intelectuais não precisarão de apoio tecnológico uma vez enraizados numa mente que cresce ativamente (PAPERT, 1985, p.23).

O movimento iniciado por Papert na década de 1960 é revitalizado (modernizado) na proposta de Wing (2006), introduzindo o termo “Pensamento Computacional” (*Computational Thinking)* e mostra-se uma tendência em ascensão desde então. Segundo a autora, o pensamento computacional envolve a solução de problemas, a criação de sistemas e a compreensão do comportamento humano, com base nos conceitos fundamentais da ciência da computação. Nesse caso, pensar de maneira computacional mobiliza uma gama de ferramentas mentais que refletem o alcance do campo de estudo da ciência da computação (WING, 2006).

Sobre essa temática Brackmann (2017, p.25) destaca que “o termo ‘Pensamento Computacional’ jamais pode ser confundido com a simples aptidão de manusear aplicativos em dispositivos eletrônicos (Alfabetismo Digital) ou uma forma de pensar de forma mecânica, limitando a criatividade da mente humana”. Nesse

caso, a proposta desse movimento orbita na intencionalidade de promover a flexibilização da maneira de pensar – nunca de engessá-la. Da mesma forma Valente (2016) comenta que ao incorporar nas escolas atividades que envolvam os conceitos de computação, espera-se que os estudantes desenvolvam habilidades do pensamento crítico e computacional, que os permita entender como criar com as tecnologias digitais, “e não simplesmente utilizá-las como máquinas de escritório” (p.867).

Frente a essa tendência muitos países vêm adequando seus sistemas de ensino, pela incorporação de disciplina voltada ao ensino de computação desde os primeiros anos da educação básica e, em alguns casos, desde a pré-escola. É o que aponta o estudo realizado por Brackmann (2017) indicando que instituições governamentais e privadas da Europa já iniciaram esse movimento. Segundo o autor o aprendizado de noções de Computação em escolas da educação básica é, atualmente, uma preocupação em diversos países, sendo que a sua “implantação ocorre de forma rigorosa” acreditando que a “Ciência da Computação é uma competência tão importante quanto ler, escrever e fazer cálculos” (BRACKMANN, 2017, p.83).

O autor destaca também que a disciplina de Computação é bastante distinta das aulas de Informática, pois agrega conhecimento e habilidades próprias da ciência que dá origem a ela. Nesse caso, estudar computação possui benefícios educacionais e econômicos, visto que, pelo viés educativo, exercitamos “habilidades de reflexão e solução de problemas, compreensão de que o mundo está impregnado com a tecnologia digital” e, de outro lado, poderemos suprir a alta demanda do mercado profissional em tecnologia com a formação de profissionais competentes (BRACKMANN, 2017).

Diante de todos os elementos citados nessa seção, acerca da importância do estudo de programação de computadores e da estimulação do pensamento computacional - seja pelo preenchimento da demanda profissional no ramo da tecnologia ou pela democratização das habilidades cognitivas indispensáveis às gerações vindouras - é que se pode considerar fundamental o ensino e aprendizagem de conceitos de computação em qualquer curso da educação básica, sobretudo, nos cursos profissionalizantes da área da Informação e Comunicação.

2.4 APRENDIZAGEM NA ATUALIDADE E O USO DE JOGOS NA EDUCAÇÃO

Na atualidade, é consenso mundial a necessidade de se desenvolver estratégias pedagógicas que abordem o uso de tecnologias. Segundo Shimohara e Sobreira (2015), a produção de uma aprendizagem de qualidade que vá ao encontro dos desafios e necessidades da sociedade do século XXI requer a integração de recursos tecnológicos aos processos educacionais. Tal situação é proporcionada pela expansão das tecnologias digitais no cotidiano das pessoas, com efeito direto no âmbito educacional.

É notável a influência das Tecnologias da Informação e da Comunicação (TIC) nos avanços da humanidade, possibilitando o surgimento constante de tendências econômicas, culturais e sociais (CASTELLS, 2016). Tais tendências impactam nos mais diversos espaços da convivência humana, com reflexo nos processos educativos. Segundo Coll e Monereo (2010), as TIC foram amplamente estudadas por diversos autores, de diversas áreas, os quais constatam em consenso genérico, que a sua evolução ocorre primeiramente pelo viés da comunicação, com destaque a três etapas-chave:

I. Primeiro a linguagem natural que possibilitou que o homem pudesse se comunicar e, dessa maneira, se adaptar ao território hostil em que vivia através da cooperação;
II. A segunda etapa consiste na hegemonia do humano sobre as demais espécies pelo domínio de técnicas capazes, em um primeiro momento, de lhe proporcionar alimento e proteção. Todavia, Coll e Monereo (2010) destacam que o aperfeiçoamento da técnica lhes proporcionou notáveis avanços nas ciências, sobretudo, no ramo da tecnologia;
III. Por fim, a terceira etapa consiste, segundo Coll e Monereo (2010, p.21), “na utilização do adjetivo ‘virtual’ para referir-se às organizações, comunidades, atividades e práticas que operam e ocorrem na internet”. Desse modo a internet pode operar como imitadora da realidade, criando outros espaços de convivência e interação profissional, social ou educacional.

Todavia, o foco desse trabalho converge para a etapa contemporânea desse processo, na qual reside o movimento de aparelhamento dos espaços educacionais, organizados e frequentados por atores imersos no contexto tecnológico em que aparelhos eletrônicos são de uso comum e as relações sociais concorrem entre o presencial e o virtual (LÉVY, 1996). Essa dinâmica atual das relações sociais

desencadeia constantes mudanças nos perfis e comportamentos da sociedade global com reflexo direto na forma como jovens compreendem o contexto em que vivem.

Em consonância com essa colocação Veen e Vrakking (2009), descrevem a característica dos jovens da atualidade, imersos nas tecnologias digitais, lidando diariamente – e com extrema naturalidade - com um complexo rizoma de informações e redes de contato, tratando com rapidez o que lhes é importante e desprezando fortuitamente o que não lhes interessa. Segundo os autores, o uso das tecnologias, sobretudo as digitais, é característico da geração da qual batizaram de "*Homo Zappiens*" pelo comportamento pautado na realização concorrente de diferentes tarefas diárias, alternando constantemente ("*zapeando"*) entre elas.

Outro aspecto fundamental dessa geração é relacionado ao imediatismo, verificado na necessidade que possuem de realizar rapidamente cada uma das tarefas, sendo que ficam rapidamente entediadas caso necessitem conduzir mais de uma tentativa para solucionar um problema. Nesse caso, dificilmente ficam tentando abstrair possíveis soluções de maneira solitária, pois são aparelhados de uma vasta rede (virtual) de amigos (*online* em tempo integral), a quem recorrem sempre que necessário. Diferente das gerações anteriores, *Homo Zappiens*, não operam em linearidade, costumam experimentar as sensações e aprender com elas, ao invés de ler o manual e executar um passo a passo (VEEN e VRAKKING, 2009).

Desse modo, Veen e Wrakking (2009), Mattar (2009), Coll e Monereo (2010), Prensky (2001, 2012) e vários outros autores dedicam especial atenção para a mudança de paradigmas na educação, ocasionada, sobretudo, pela influência e uso das tecnologias digitais. De todo modo, ambos os autores citados concordam que a mudança de comportamento dos estudantes evolui em uma esfera global ocasionada pela disseminação das tecnologias em rede.

Contudo, o estudante da atualidade, atravessado pelo impacto da Sociedade da Informação (SI) (COLL e MONEREO, 2010) é sujeito humano, afetado, desse modo, por questões de ordem genética e do convívio diário em suas interações cotidianas. Assim, embora partilhe dos mesmos hábitos de seus pares, em relação à forma de aprender, desenvolve o seu próprio perfil o qual é estruturado de acordo com seu contexto evolutivo particular. Tal situação é vislumbrada diariamente nas escolas, pois, dinâmicas de ensino adotadas pelos docentes, em geral, não atingem uma turma em sua totalidade (VEEN e WRAKKING, 2009; COLL e MONEREO, 2010).

Dessa maneira, para atender a presente demanda educacional com qualidade, a busca por formas alternativas de ensino é sempre uma constante em todos os níveis escolares. Em cursos de computação isso não é diferente, pois possuem uma característica muito peculiar no que tange à grande quantidade de disciplinas técnicas, as quais abordam complexos conceitos da área (WANGENHEIM e WANGENHEIM, 2012).

Assim, se de um lado temos a complexidade dos conteúdos estudados na área da computação, de outro temos as rápidas transformações verificadas nos perfis dos estudantes da contemporaneidade, produtos da SI a qual comporta novas formas de trabalhar, de comunicar-se, de aprender e de pensar (COLL e MONEREO, 2010; CASTELLS, 2016). Desse modo, o empenho pelo desenvolvimento e utilização de formas mais eficientes para ensinar e aprender computação entra em pauta de discussão entre professores da área.

Muitas são as técnicas, metodologias e abordagens pedagógicas verificadas na atualidade, cada uma delas convergindo para necessidade pontuais dos estudantes. Desse modo, Wangenheim e Wangenheim (2012) destacam cinco métodos instrucionais, quais sejam: instrução direta; instrução indireta; instrução interativa; estudo individual e experiencial (Figura 6).

Figura 6 - Métodos instrucionais

Fonte: Wangenheim e Wangenheim (2012).

Todavia, para esse trabalho, a atenção especial é designada para o uso de jogos como ferramenta potencializadora do processo educativo. A escolha se deve pela grande variedade de possibilidades de uso e pela sua popularidade frente ao público alvo dessa pesquisa. Desse modo, faz-se necessário entender a dinâmica envolvida no contexto da sua utilização para o viés educativo.

Nesse sentido, muitas vezes nos perguntamos: *Mas por que jogos são tão populares?* A resposta para essa pergunta pode ser encontrada no fator motivacional imbricado ao seu uso. Todavia, Johan Huizinga vai além desse contexto explicando que no jogo residem fatores culturais que influenciam na formação daquele que joga, visto que tantas outras atividades humanas são atravessadas por eles – linguagens, conhecimento, poesia, arte, filosofia, etc. Assim, o "*Jogo*" é, para além da brincadeira, uma atividade constitutiva do sujeito (HUIZINGA, 2007).

Do mesmo modo, Mattar (2009) considera, para além dos elementos já expostos, outros aspectos relacionados à popularidade dos jogos, destacando também, a possibilidade de o jogador assumir diferentes papéis e identidades virtuais.

> Por meio dos videogames as crianças aprendem, por exemplo, a brincar com identidades, assumindo e construindo diferentes personalidades virtuais. Em muitos casos, essas personalidades envolvem a identidade de solucionador de problemas, mediante a qual a criança aprende a lidar com erros de uma forma mais dinâmica e interativa do que ocorre na escola (MATTAR, 2009, p. 31).

Tais colocações corroboram com Huizinga (2007), que destaca a capacidade lúdica do jogo, a possibilidade de tratar os jogadores como atores de um cenário impossível de se produzir no mundo real, onde a imaginação ganha espaço podendo, se necessário, desprezar o mundo físico como o conhecemos.

Prensky (2012), considerado por muitos uma referência na aprendizagem baseada em jogos, destaca que o ensino tradicional é baseado na realização de tarefas, caracterizando isso como "trabalho pesado" e, portanto, "chato", cansativo e tedioso. De outro lado, ao introduzir a dinâmica de jogos, a aprendizagem ganha um viés de ludicidade, proporcionando prazer em realizar a atividade. Esse movimento de prazer ao realizar a tarefa pode ser ilustrado pelo que Prensky (2012) destaca ao citar Doug CrockFord[11] como "aprendizagem disfarçada", pois o estudante aprende enquanto está envolvido na realização da tarefa.

[11] Douglas trabalhou como programador no Atari, Lucasfilm e Paramount. Tornou-se referência na temática de vídeogames nos anos 1990 depois de ter colocado suas memórias no jogo "The Expurgation of Maniac Mansion".

As contribuições de Prensky (2012) são reforçadas pelas ideias de Wangenheim e Wangenheim (2012) ao descrever a pouca eficiência de aulas expositivas para o público jovem da atualidade, rotulado pelos autores de “geração y”:

> Além disso, aulas expositivas são consideradas chatas e não ajudam a motivar o aluno a aprender. Isso, principalmente levando em consideração, que nossos alunos de hoje – representados pela geração Y – são acostumados a se comportar de forma intensamente interativa e esperam um retorno imediato (WANGENHEIM e WANGENHEIM, 2012, p.15).

De todo modo, Prensky (2012) considera ainda que o sucesso da aprendizagem baseada em jogos digitais se apoia em duas premissas radicais relacionadas ao perfil dos aprendizes da contemporaneidade. Nesse caso, o autor define que a primeira premissa “radical” está na mudança em pontos essenciais dos aprendizes, no que se refere a “como aprendem”. Nesse caso, é possível atribuir essa mudança ao panorama tecnológico, a partir da disseminação das mídias de informação e democratização das tecnologias. A segunda premissa está relacionada com a mudança que a geração dita “Nativa Digital[12]” desenvolveu na forma de brincar, pois em geral essa geração modificou drasticamente as brincadeiras da infância voltando-as para o uso de videogames e computadores.

Sobre os benefícios do uso de jogos em favor da aprendizagem, Coll e Monereo (2010) destacam que diversos estudos realizados acerca do impacto dos jogos nas áreas cognitivas apontaram que, a partir do seu uso, vários elementos são favorecidos, com destaque à “atenção visual, a representação espacial, o descobrimento indutivo, a representação icônica e a construção de gênero”. Os autores destacam também que as habilidades favorecidas pelo uso de jogos convergem para a melhoria da vida dos indivíduos, pois “são úteis para o manejo de diversas ferramentas envolvidas em práticas socais emergentes” (COLL e MONEREO, 2010, p.52).

Frente a todos esses fatores, jogos comportam vários elementos característicos dos jovens da atualidade, pois agregam em suas dinâmicas elementos capazes de produzir aprendizado, trazendo vários benefícios para o processo de ensino. Dentre tais benefícios Prensky (2012) destaca que é possível combinar videogames e jogos

[12] Nativos digitais é um rótulo que tem sido amplamente disseminado tanto na literatura acadêmica quanto nas grandes mídias, em discussões acerca dos usos de TIC por jovens, em particular, os usos de artefatos Web 2.0 (FERREIRA e CASTIGLIONE, 2017).

de computador com uma grande variedade de conteúdos educacionais, atingindo resultados melhores que as metodologias tradicionais de ensino.

Do mesmo modo, Wangenheim e Wangenheim (2012) apontam que a partir de alguns tipos de jogos é possível oferecer um ambiente seguro em que o aluno possa experimentar diversas alternativas e visualizar as consequências. Nesse mesmo viés, os autores destacam que jogos também possibilitam ao aluno aprender com os erros e lhes proporcionam um *feedback* instantâneo de suas interações.

A escolha da dinâmica de jogos para esse trabalho está relacionada aos benefícios e possibilidades elencados acima, porém, aprimora-se a ideia de aprendizagem baseada em jogos para, nesse caso em específico, para aprendizagem baseada *no desenvolvimento de jogos*. De todo modo, considera-se que os benefícios relacionados à aprendizagem por jogos são mantidos, pois as premissas elencadas acima também podem ser aplicadas para a condição de *desenvolvimento*.

É importante salientar que, no caso do ensino de programação de computadores, utilizar a perspectiva do desenvolvimento de jogos, pode trazer outros benefícios, pois, possibilita também que o estudante possa aprender fazendo, reduzindo assim a lacuna existente entre teoria e prática, além de proporcionar maior envolvimento do aprendiz com o conteúdo (WANGENHEIM e WANGENHEIM, 2012).

No próximo capítulo, é apresentada uma revisão sistemática da literatura, que teve como finalidade pesquisar, em nível nacional, quais são as principais metodologias e ferramentas atualmente utilizadas para o ensino de programação de computadores, com ênfase em POO.

3 O ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS COM A UTILIZAÇÃO DE *SOFTWARES* EDUCACIONAIS: UMA REVISÃO SISTEMÁTICA DE LITERATURA

As metodologias de ensino e aprendizagem nunca estiveram tão dinâmicas como nos dias atuais. No ensino da computação, a inclusão de tais práticas busca oferecer facilitadores capazes de amenizar as dificuldades de assimilação dos conceitos relacionados à lógica computacional. A complexidade relacionada a tais conceitos se deve pelo fato da necessidade de dominar diversas áreas do conhecimento, que perpassam desde a interpretação do problema, o desenvolvimento prático de uma solução e a construção sintática de um algoritmo para tal. Nesse caso, a solução de problemas computacionais desencadeia processos mentais que necessitam da mobilização de habilidades intelectuais diversas.

O domínio dessas habilidades é parte integrante do rol de significados necessários à convivência moderna, em que os mecanismos digitais são indispensáveis às atividades cotidianas. Essa tendência é observada há várias décadas e, em especial, Seymour Papert e Wally Feurzeig, ao desenvolver a linguagem LOGO no final dos anos 1960, na busca pela utilização do computador em favor da construção do conhecimento e da estimulação do pensamento computacional já nas séries iniciais do ensino fundamental.

Muitos anos se passaram desde o início desse movimento e, na atualidade, ainda encontramos sérios desafios no ensino da programação de computadores, evidenciados em várias publicações que demonstram que a complexidade dessa área do conhecimento tem ocasionado muitas desistências em diversos cursos no Brasil. Segundo Jesus, Gonçalves e Ferreira (2014) as escolas e universidades vêm sofrendo um esvaziamento nos cursos da área de computação tendo em vista a dificuldade dos alunos em compreender a lógica de programação.

Nesse contexto, o presente capítulo tem como objetivo apresentar uma Revisão Sistemática da Literatura (RSL) utilizando a abordagem proposta por Kitchenham (2007), realizada com o objetivo de investigar e identificar quais são as principais ferramentas e referenciais metodológicos utilizados no ensino da programação de computadores, com ênfase em OO. O direcionamento da pesquisa buscou identificar,

em especial, os processos didáticos para o ensino de POO envolvendo o uso de tecnologias.

Nesse sentido, o trabalho visou elencar, filtrar e analisar as publicações na área, incluindo artigos publicados no período de 2010 a 2016, nos principais periódicos e anais de eventos da área de Informática na Educação no Brasil. Uma RSL visa pesquisar em profundidade um objeto de interesse e produz resultados detalhados por meio da análise de conteúdo e qualidade do material pesquisado (KITCHENHAM, 2007).

3.1 METODOLOGIA DA RSL

Kitchenham (2007) define as diretrizes para a elaboração da RSL a partir da realização de uma sequência de passos que permitem sistematizar todo o processo de revisão. Tal sequência perpassa, primeiramente, pela definição inicial do propósito da pesquisa, o âmbito e o período temporal a ser estudado. Na fase seguinte é realizada a busca de publicações em fontes de pesquisa determinadas pelo âmbito selecionado. A partir dos resultados brutos dessa fase é necessário estabelecer critérios de inclusão e exclusão para que sejam selecionadas apenas publicações com potencial para elucidar o objeto de pesquisa. O passo seguinte versa sobre a extração dos dados mais relevantes para a pesquisa. Assim, é necessário estipular critérios de avaliação da qualidade das publicações selecionadas a partir da verificação de seus conteúdos. Por fim, chega-se a parte da análise, em que são tabulados os resultados dos cruzamentos das perguntas de extração. Tal cruzamento é capaz de oferecer as informações necessárias para sintetizar o “estado da arte” do objeto de pesquisa, auxiliando o pesquisador na tomada de decisões quanto ao futuro da pesquisa na área.

3.2 OBJETIVO E PROCESSO DE BUSCA

Esta revisão teve como objetivo identificar quais são as principais ferramentas e metodologias utilizadas no ensino de programação, realizando uma análise do panorama das pesquisas e experiências no âmbito nacional. Nesse sentido, o trabalho

visou elencar, filtrar e analisar as publicações na área, incluindo artigos publicados entre o período de 1º de janeiro de 2010 a 31 de dezembro de 2016. Para essa pesquisa, foram relacionados artigos publicados nos anais dos seguintes eventos:

a) Simpósio Brasileiro de Informática na Educação (SBIE);
b) Workshop de Informática na Escola (WIE);
c) Revista Brasileira de Informática na Educação (RBIE);
d) Revista Novas Tecnologias na Educação (RENOTE);
e) Workshop de Ensino em Pensamento Computacional, Algoritmos e Programação (WAlgProg);
f) Congresso Alice Brasil (Alice).

Os eventos SBIE, WIE, RBIE, WAlgProg e RENOTE foram selecionados com base na sua abrangência nacional e pelo seu cunho voltado para o uso da informática educativa. Já o evento Alice Brasil não é tão conhecido da comunidade, mas destina-se especificamente a publicar experiências relacionadas ao ensino da programação a partir do uso do *software* ALICE o qual dá destaque ao paradigma de POO.

A RSL visa a responder às questões apresentadas na Tabela 1:

Tabela 1 - Questões de Pesquisa.

Questões	Descrição
QP1	Que estratégias ou métodos têm sido utilizadas para o ensino da programação?
QP2	Quais são os principais *software*s utilizados para mediar este processo?
QP3	Quais benefícios e limitações estão sendo relatados pelo uso das abordagens propostas para o ensino de POO?

Fonte: do autor.

3.3 STRING DE BUSCA E ESTUDOS PRIMÁRIOS

Definidas as questões da pesquisa, procedeu-se a construção de uma *string* de busca para obtenção de resultados primários dessa RSL. A mesma *string* é utilizada nos motores de busca dos repositórios de dados dos eventos pesquisados. As palavras-chave utilizadas para a sua composição foram: metodologia, ensino e programação. Com base neste conjunto de palavras, a *string* de busca construída foi:

((ferramenta OR software OR estratégias OR abordagens OR práticas OR metodologias OR métodos) AND ensino AND programação).

É importante salientar que a primeira parte da *string* busca incluir possíveis variantes da palavra-chave “Metodologia” e suas flexões gramaticais. Já a segunda parte determina que os resultados contenham, necessariamente, as palavras “Ensino” e “Programação”.

A RSL foi realizada em três etapas. A primeira consistiu na busca por publicações nos motores de busca dos eventos SBIE, WIE, RBIE e RENOTE, os quais, juntos retornaram 86 resultados. No caso dos anais do evento Alice Brasil e WAlgProg foram elencadas todas as publicações do período pesquisado, visto que não há motor de busca em que se possa aplicar processo automatizado. Nesse caso, foram incluídas mais 74 publicações dos referidos eventos. Assim, um total de 160 publicações compuseram os estudos primários.

3.4 SELEÇÃO DOS DADOS

A partir da seleção inicial das publicações, é realizada a leitura dos títulos, resumos e palavras-chave. Esse processo se faz necessário para realizar uma triagem inicial dos trabalhos potenciais para a pesquisa, bem como a exclusão das publicações que não abordam questões relevantes para esta revisão. Nessa fase, foram definidos e aplicados Critérios de Inclusão (CI) e Critérios de Exclusão (CE) para cada estudo. A Tabela 2 relaciona tais critérios:

Tabela 2 - Critérios de inclusão e exclusão

Critérios de Inclusão	Critérios de Exclusão
CI1: É uma ou método de ensino de programação;	CE1: Não está em português ou inglês;
CI2: Usa uma ferramenta especifica;	CE2: Anterior a 2010;
CI3: Aborda jogos para o ensino da computação;	CE3: Resumos de anais;
CI4: Revisão da literatura acerca do ensino da programação;	CE4: Não atende aos propósitos da pesquisa

Fonte: do autor.

A partir dessa primeira triagem, dos 160 estudos, foram selecionados 86 trabalhos (53%). Outros 72 (45%) foram rejeitados e 2 (1%) são duplicados. Dos

trabalhos rejeitados, 14 (19%) foram excluídos pelo critério de exclusão CE2 e 58 (81%) pelo critério CE4.

Seguindo os critérios de inclusão dos trabalhos, essa primeira análise nos oferece os seguintes quantitativos: CI1 foi verificado em 72 estudos (83%), o CI2 aparece em 63(73%), o CI3 está presente em 14 (16%) estudos e, por fim, o CI4 foi verificado em 13(16%) das publicações pré-selecionadas.

3.5 EXTRAÇÃO DOS DADOS E AVALIAÇÃO DA QUALIDADE

De posse dos estudos pré-selecionados, faz-se necessário a extração dos dados mais relevante à RSL. Nesse caso, com base nas diretrizes propostas por Kitchenham (2007), devemos estabelecer critérios de avaliação que possibilitem a análise e extração de conteúdo significativo à proposta da RSL. Nessa fase, os estudos devem ser lidos na íntegra, a fim de buscar subsídios capazes de responder às questões definidas nos critérios de extração. A Tabela 3 elenca os critérios utilizados.

Durante essa fase, é possível ainda, a partir da leitura mais aprofundada dos trabalhos selecionados, rejeitar àqueles cujo conteúdo não atenda aos propósitos da RSL. Nesse sentido, a partir da realização das leituras foi possível rejeitar 20 publicações, cujo conteúdo não atende aos objetivos dessa pesquisa. Nesse caso, permaneceram para a RSL 66 estudos elencados na Tabela 4.

Tabela 3 – Critérios de Extração

Critério	Descrição
CEXT1	Foi avaliado em sala de aula?
CEXT2	Para que faixa etária foi desenvolvido?
CEXT3	Qual é o nível de programação?
CEXT4	Qual *software* foi proposto ou utilizado?
CEXT5	Utilizou alguma metodologia/Abordagem de desenvolvimento? Qual?
CEXT6	Qual abordagem pedagógica é encontrada?

Fonte: do autor.

O CEXT1 busca selecionar trabalhos cujas práticas de ensino tenham sido efetivamente conduzidas em salas de aula. Todavia, vale salientar que a opção

negativa deste critério não rejeita da revisão, pois pode conter informações relevantes aos demais objetivos desse estudo.

Para obter informações quanto ao nível de ensino que os estudos estão sendo realizados, o CEXT2 busca classificá-los nos níveis: “Ensino fundamental”, “Ensino Médio”, “Ensino Superior” ou “Outro”. Da mesma forma, o CEXT3 foi incluído com a finalidade de mapear para que níveis de complexidade da área da programação tais práticas têm sido desenvolvidas, oferecendo quantitativos relacionados ao número de estudos voltados ao ensino de “Algoritmos ou Lógica de programação”, “Programação Estruturada” ou “Programação Orientada a Objetos”.

Para buscar informações mais diretas em relação aos *software*s utilizados para o ensino de programação, o critério CEXT4 tem a finalidade de agrupar quantitativos sobre as principais ferramentas utilizadas para apoiar tais práticas. Nesse caso, a intenção é mapear cada uma das ferramentas citadas nos estudos pesquisados, para que se conheça também, para além dos quantitativos, a maior variedade possível de ferramentas para o ensino de programação.

Os critérios CEXT5 e CEXT6 buscam selecionar aspectos pedagógicos envolvidos nas práticas de ensino relatadas, sendo que o primeiro dedica atenção especial no sentido de verificar quais *estratégias pedagógicas* estão sendo utilizadas; já o CEXT6 tem a finalidade de verificar as *abordagens pedagógicas* evidenciadas nos estudos selecionados.

Tabela 4 - Artigos incluídos

ID	Autor	Título	Ano
1	Souza, Batista e Barbosa	Problemas e Dificuldades no Ensino de Programação: Um Mapeamento Sistemático	2016
2	Wangenheim, Nunes, e Santos	Ensino de Computação com SCRATCH no Ensino Fundamental – Um Estudo de Caso	2014
3	Scaico et al	Ensino de Programação no Ensino Médio: Uma Abordagem Orientada ao Design com a linguagem Scratch	2013
4	Batista et al	Utilizando o Scratch como ferramenta de apoio para desenvolver o raciocínio lógico das crianças do ensino básico de uma forma multidisciplinar	2015
5	Santos, Cristiano e Mota Neto	Raciocínio Lógico e Computação: Descobrindo Estratégias de ensino por meio da Olimpíada Brasileira de Informática	2015
6	Cambruzzi e Souza	Robótica Educativa na aprendizagem de Lógica de Programação: Aplicação e análise.	2015
7	Teixeira et al	Programação de computadores para alunos do ensino fundamental: A Escola de Hackers	2015
8	Souza e Mombach	Ensino de Programação para Crianças através de Práticas Colaborativas nas Escolas	2016
9	Oro et al	Olimpíada de Programação de Computadores para Estudantes do Ensino Fundamental: A interdisciplinaridade por meio do *software* Scratch	2015
10	Ribeiro, Bazílio Martins e Bernardini	A Robótica como Ferramenta de Apoio ao Ensino de Disciplinas de Programação em Cursos de Computação e Engenharia	2011
11	Rafalski e Santos	Uma experiência com a Linguagem Scratch no Ensino de Programação com Alunos do Curso de Engenharia Elétrica	2016
12	Marques et al	Atraindo Alunos do Ensino Médio para a Computação: Uma Experiência Prática de Introdução a Programação utilizando Jogos e Python	2011

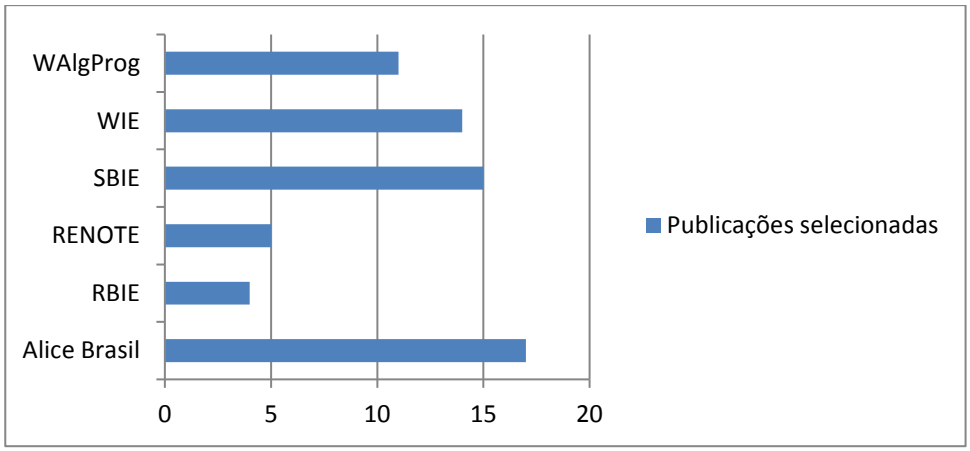
13	Jesus, Gonçalves e Ferreira	Aplicação de Desenvolvimento de Jogos Digitais como um Meio de Motivação em Diferentes Níveis de Ensino de Computação	2014
14	Thomaz et al	RoboEduc: Um *software* para Programação em Níveis	2010
15	Ramos e Teixeira	Significação da Aprendizagem Através do Pensamento Computacional no Ensino Médio: uma Experiência com Scratch	2015
16	Monteiro et al	Uma Experiência do Uso Do Hardware Livre Arduino no Ensino De Programação De Computadores	2016
17	Finizola et al	O ensino de programação para dispositivos móveis utilizando o MIT-App Inventor com alunos do ensino médio	2014
18	Heinen et al	RASPIBLOCOS: Ambiente de Programação Didático Baseado em Raspberry Pi e Blockly	2015
19	Souza et al	Lord of Code: Uma Ferramenta de Apoio ao Ensino da Programação	2016
20	Galvão, Fernandes e Gadelha	Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação	2016
21	Alencar et al	RHODES 2.0: *software* Educacional para o Ensino de Programação Linear	2012
22	Scaico e Scaico	Uso de Jogos em Cursos Introdutórios de Programação - Uma Revisão Sistemática	2016
23	França e Tedesco	Caracterizando a pesquisa sobre autoavaliação na aprendizagem de programação para iniciantes	2015
24	Mattos, Ferreira e Anacleto	O Ensino de Programação com Scratch e seu Impacto na Opção Profissional para Meninas	2016
25	Ribeiro, Brandão e Brandão	Uma visão do cenário Nacional do Ensino de Algoritmos e Programação: uma proposta baseada no Paradigma de Programção Visual	2012
26	Marcolino e Barbosa	*software*s Educacionais para o Ensino de Programação: Um Mapeamento Sistemático	2015
27	Medeiros, Brasil e Aranha	Um framework para criação de jogos voltados para o ensino de lógica de programação	2014
28	Belchior, Bonifácio e Ferreira	Avaliando o Uso da Ferramenta Scratch para Ensino de Programação através de Análise Quantitativa e Qualitativa	2015
29	Mota et al	Desenvolvendo o Raciocínio Lógico no Ensino Médio: uma proposta utilizando a ferramenta Scratch	2014
30	Zanetti e Bonacin	Uma Metodologia Baseada em Semiótica para Elaboração e Análise de Práticas de Ensino de Programação com Robótica Pedagógica	2014
31	Silva, Medeiros e Aranha	Jogos Digitais para Ensino e Aprendizagem de Programação: uma Revisão Sistemática da Literatura	2014
32	Amaral, Braga e Silva e Pantaleão	Plataforma Robocode como Ferramenta Lúdica de Ensino de Programação de Computadores - Pesquisa e Extensão Universitária em Escolas Públicas de Minas Gerais	2015
33	Fernandez et al	Uma proposta baseada em projetos para oficinas de Internet das Coisas com Arduino voltadas a estudantes do Ensino Médio	2015
34	Souza, Jaeger e Cardoso	Ensino de algoritmos apoiado pelo uso de jogos digitais educativos	2013
35	Gomes e Amara	Simplificando a depuração de códigos na Linguagem C - Uma solução para alunos iniciantes	2016
36	Sousa e Favero	Resolução de problemas de programação com o método de composição de resultado	2015
37	Rocha et al	Ensino e Aprendizagem de Programação: Análise da Aplicação de Proposta Metodológica Baseada no Sistema Personalizado de Ensino	2010
38	Pedrosa e Pamboukian	ALICE no ensino de computação para crianças	2010
39	Antonio e Ferro	Avaliação do *software* Alice 3D para o Ensino de Orientação a Objetos apoiada por Ambientes Virtuais, PBL e Mapas Conceituais	2011
40	Craveiro et al	Utilizando o *software* ALICE como apoio para o Ensino de Lógica de Programação Parte 1 Estruturas Básicas	2011
41	Craveiro et al	Utilizando o *software* ALICE como apoio para o Ensino de Lógica de Programação parte 2 programação estruturada	2011
42	Craveiro et al	Utilizando o *software* ALICE como apoio para o Ensino de Lógica de Programação Parte 3 Programação Orientadas a Objetos	2011
43	Ferreira	Animalice – Festival de Animação do Alice: o uso do *software* Alice como ferramenta de aprendizagem na disciplina de robótica para estudantes do Ensino Fundamental	2012
44	Antonio e Ferro	O uso do Sofiware Alice 3D como Motivação para o Ensino-Aprendizado de Programação: Exemplos de Atividades para o Curso de Sistemas de Informação	2013
45	Schefer	Uma proposta para inclusão de lógica de programação no currículo do Ensino Médio	2013
46	Cipriano, Schefe, e Carrie	Uma abordagem pelo *software* Alice para exercícios de lógica de programação	2013
47	Martinelli e Ferraz Junior	Desenvolvimento do Raciocínio Lógico e de Jogos Eletrônicos Educativos pelo Scratch	2013
48	Zotovici	Literatura brasileira, interpretação, animação e modelagem orientada a objetos: atividades para fixação de aprendizagem em programação orientado a objetos	2013

49	Barbosa, Santos e Cruz	Projeto Investindo em Novos Talentos: Introdução à programação para crianças com o Alice 3D	2016
50	Souza et al	Uso do Scratch e do Alice para o ensino de lógica na rede pública de ensino do Litoral Norte do Estado de São Paulo	2016
51	Andrade, Moreno e Rossi	Experiência de uso do sofiware Alice em uma atividade de análise, projeto e desenvolvimento de sistemas aplicando os conceitos basicos de orientaçao a objetos	2016
52	Moraes	Gamificação e Alice: Desenvolvimento de Jogos Educativos em Oficina Escolar	2016
53	Vilela, Tsan Hu e Barros	Aprendendo estruturas de controle e repetição com o Alice	2016
54	Valletta	A hora do código: um estudo de caso no Ensino Médio	2016
55	Zanetti e Oliveira	Práticas de ensino de Programação de Computadores com Robótica Pedagógica e aplicação de Pensamento Computacional	2015
56	Mattos et al	Introdução à Robótica e Estímulo à Lógica de Programação no Ensino Básico Utilizando o Kit Educativo LEGO® Mindstorms	2015
57	Piva Jr. E Cortelazzo	Sala de aula invertida, ambientes de aprendizagem e educação online: a junção de três métodos para potencialização do ensino de algoritmos	2015
58	Cordenonzi et al	Uma Experiência Interdisciplinar no Ensino de Algoritmos e Matemática em um Contexto Binacional.	2015
59	Maekawa, Nagai e Izeki	Relato de Gamificação da disciplina Projeto e Análise de Algoritmos do curso de Engenharia de Computação	2015
60	Silva, Souza e Silva	Aplicação da Ferramenta Scratch para o Aprendizado de Programação no Ensino Fundamental I	2015
61	Souza e Castro	Investigação em programação com Scratch para crianças: uma revisão sistemática da literatura	2015
62	Kutzke e Direne	Mediação do erro no ensino de programação de computadores: fundamentos e aplicação da ferramenta FARMA-ALG	2015
63	Andrade et al	Uma Proposta de Oficina de Desenvolvimento de Jogos Digitais para Ensino de Programação	2015
64	Gomes e Amaral	Uma Proposta de Ferramenta para Simplificar a Depuração de Códigos em C, por Alunos Iniciantes	2015
65	Moreira et al	Atraindo Meninas para a Ciência da Computação: Métodos e Ferramentas	2015
66	Silva et al	Ensino-aprendizagem de programação: uma revisão sistemática da literatura	2015

Fonte: do autor.

O Gráfico 1 relaciona a distribuição das publicações entre os eventos pesquisados:

Gráfico 1 - Publicações selecionadas

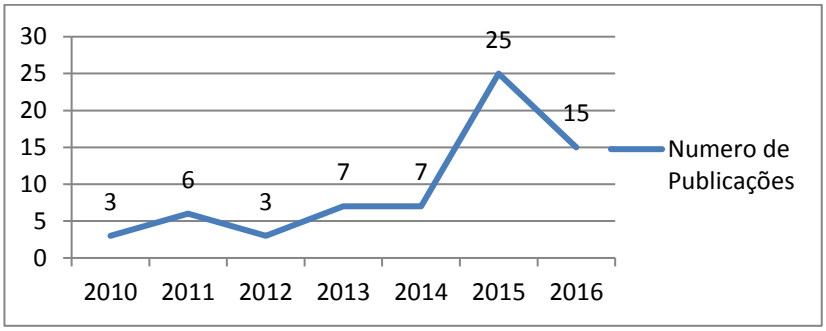


Fonte: do autor.

3.6 RESULTADOS E DISCUSSÕES

Partindo para uma análise detalhada acerca dos estudos selecionados, verificou-se que houve aumento considerável nas publicações da área nos últimos dois anos, conforme ilustrado pelo Gráfico 2, o que caracteriza que há um movimento ascendente em pesquisas a nível nacional para o estudo e implementação de práticas de ensino de programação.

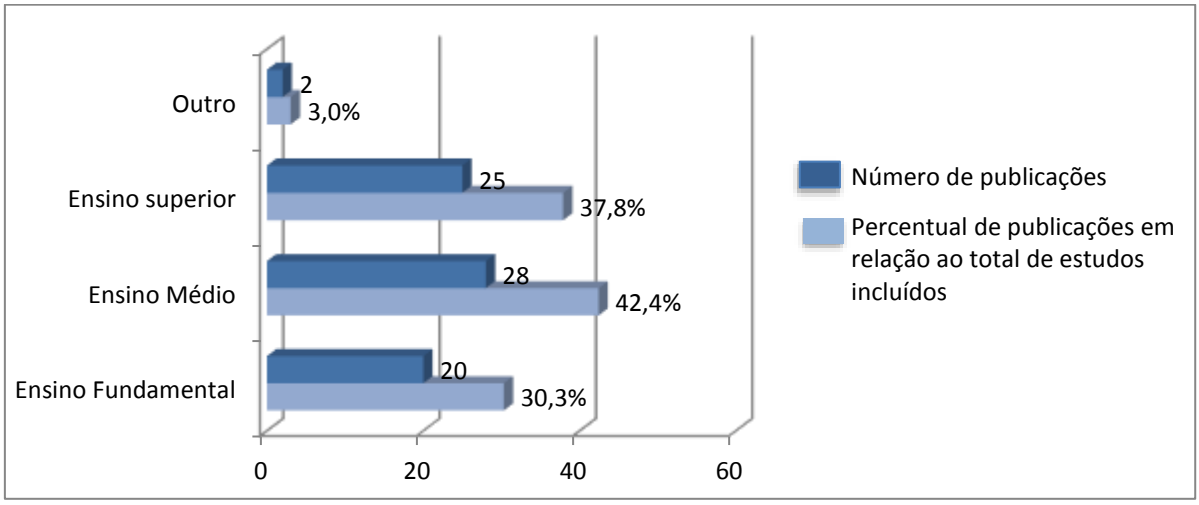
Gráfico 2 - Número de Publicações por ano



Fonte: do autor.

Constatou-se também que 46 trabalhos (70%) foram efetivamente testados em atividades práticas de ensino e apenas 20(30%) se tratam de projetos ou estudos teóricos acerca dessa temática. O Gráfico 3 ilustra a distribuição dos estudos selecionados em relação ao nível de ensino.

Gráfico 3 - Distribuição dos estudos por nível de ensino



Fonte: do autor.

Verificou-se também, que a grande maioria dos trabalhos publicados está relacionada à introdução à programação, pois, 85% dos estudos analisados foram

assinalados com a opção “Algoritmos ou Lógica de programação”. A “Programação Estruturada” foi abordada em 32% dos trabalhos e a “Programação Orientada a Objetos” em apenas 12% dos estudos analisados, sendo que vários trabalhos utilizaram abordagens em mais de um nível de programação.

A partir desses primeiros resultados – número de publicações por ano, contexto prático dos estudos, distribuição por diferentes níveis de ensino e complexidade dos conceitos - é possível iniciar uma análise qualitativa visando alcançar o propósito dessa RSL. Nesse sentido, os quantitativos obtidos indicam a existência de muitas ações e pesquisas voltadas à inclusão e ao aperfeiçoamento de práticas de ensino da programação de computadores no país.

Embora cursos de computação só existam para os níveis médio e superior, percebe-se que $^1/_3$ dos trabalhos envolveram o Ensino Fundamental. A inclusão de conceitos de computação já no ensino fundamental é uma tendência que vem agregando adeptos no campo pedagógico há vários anos. Essa linha de pensamento é fortalecida pela facilidade e interesse que as crianças têm com ambientes computacionais e o panorama tecnológico atual arraigado de opções.

Em relação ao ensino médio e superior, é que encontramos o maior quantitativo de publicações (42% e 38%, respectivamente). Diferente do ensino fundamental, no ensino médio e no superior há cursos de computação em que o ensino da programação é disciplina integrante da grade curricular de muitos cursos. Nesse sentido, boa parte dos desafios do ensino da programação emergem nesses níveis do ensino.

Conforme estudo realizado por Souza, Batista e Barbosa (2016), o ensino e a aprendizagem de programação tem sido considerados por muitos professores e alunos como uma tarefa de difícil aprendizagem e, por esse motivo, ocasiona consequências desagradáveis. Em reflexo a essa problemática, segundo os autores, muitos cursos de programação frequentemente apresentam altas taxas de reprovação e desistência. As dificuldades apresentadas pelos estudantes estão relacionadas em abstrair conceitos fundamentais da programação, tais como ponteiros, recursão e declaração de variáveis.

Ainda, alguns autores ressaltam que os estudantes conseguem entender os conceitos da programação, mas encontram dificuldades em realizar a transposição de problemas reais em programas capazes de solucioná-los. A motivação para estudar programação também é um fator a ser considerado, pois vários estudos apontam esse

quesito como ponto limitador da aprendizagem, uma vez que muitos alunos consideram a programação como uma atividade “cansativa e tediosa” (SOUZA, BATISTA e BARBOSA, 2016).

Scaico et al (2013) comentam que há no meio discente informações equivocadas sobre a carreira na área da programação, nutrindo, entre os estudantes, preconceitos negativos que influenciam no desinteresse por esse campo de estudo. Os autores salientam que a computação não é um componente explorado nos currículos escolares e, por consequência, essa situação atua como agravante no desinteresse dos estudantes pela área.

Um fator a ser considerado é a dificuldade em abstrair os problemas e construir algoritmos capazes de solucioná-los. Nesse caso são relatadas situações que vão além de problemas com a sintaxe das linguagens de programação, pois muitos autores consideram que os estudantes têm dificuldades com o raciocínio lógico, o domínio de técnicas de programação no que tange a fragmentação de problemas em problemas menores dividindo em funções, procedimentos ou classes.

3.6.1 QP1 - Que estratégias ou métodos têm sido utilizadas para o ensino da programação?

A QP1 tem como objetivo elencar e analisar as principais abordagens, metodologias e abordagens pedagógicas presentes nos estudos selecionados. Nesse sentido, para obter informações organizadas acerca das estratégias e abordagens verificadas nos estudos, foram definidas categorias que pudessem classificá-las tomando-se como base a proposta de Silva et al. (2015). Assim, a classificação de tais estratégias é apresentada na Tabela 5.

Tabela 5 – Estratégias ou métodos

Estratégia/Método	Qtd	%	Estudos
Avaliação de ferramenta	4	6%	E19, E21, E25 e E28
Metodologia*	26	39%	E4, E05, E6, E08, E09, E18, E20, E30, E32, E33, E36, E37, E39, E43, E44, E48, E49, E51, E52, E54, E56, E57, E58, E59, E60, E63
Robótica	9	13%	E06, E10, E14, E16, E18, E33, E55, E56, E60
Jogos	11	15%	E11, E12, E13, E15,E19, E31, E32, E34, E47, E59, E63

* Os trabalhos categorizados como “Metodologia”, foram enquadrados nessa categoria conforme classificação realizada pelos autores dos trabalhos pesquisados.

Fonte: do autor.

Silva et al. (2015) sugere, ainda, a existência de outras classificações, porém nos estudos pesquisados nessa revisão sistemática não foram observados em sua totalidade. Todavia, algumas classificações propostas pelos autores sugerem subclassificações na sua estrutura, as quais não serão realizadas nessa RSL.

Constatou-se que 6% dos estudos destinaram-se a realização de avaliações de ferramentas de *software*. Nesse caso, os *software*s avaliados foram: *Lord of code*, RHODES 2.0, iVProg e Scratch, com destaque para a avaliação das funcionalidades do Scratch realizada pelo estudo E28. Os autores utilizaram o método baseado na teoria da Engenharia Semiótica (EngSem) para analisar os princípios de comunicação e usabilidade da ferramenta e, para avaliar a metacomunicação transmitida do *design* para o usuário, utilizou-se o Método de Avaliação da Comunicabilidade (MAC). Os resultados indicam que o uso do Scratch é recomendado a qualquer tipo de usuário, mesmo para os que não têm muita experiência em programação.

Metodologias diversas foram relatadas em 39% dos trabalhos, conforme Tabela 6:

Tabela 6 - Metodologias relatadas

Metodologia* Descrição	Qtd	Estudos
Aprendizagem baseada em problemas (PLB)	4	E5, E18, E39 e E56
Aprendizagem por projetos	7	E33, E43, E44, E48, E49, E51 e E52
Competição/Olimpíada	2	E9 e E32
Ensino Híbrido	1	E20
Mapas conceituais	1	E39
Método de composição de resultado	1	E36
Metodologia baseada em semiótica para concepção, elaboração e análise de práticas.	1	E30
Objetos de aprendizagem	3	E4, E6 e E58
Oficinas	3	E59, E60 e E63
Práticas colaborativas	1	E8
Sala de aula invertida	2	E54 e E57
Sistema Personalizado de Ensino	1	E37

* Os trabalhos categorizados como “Metodologia”, foram enquadrados nessa categoria conforme classificação realizada pelos autores dos trabalhos pesquisados.

Fonte: do autor.

O uso da robótica e de jogos esteve presente em, respectivamente, 13% e 15% dos estudos, sendo que alguns destes apareceram em mais de uma categoria, pois foram desenvolvidos utilizando jogos ou robótica em conjunto com alguma metodologia – é o caso dos estudos E56, E59, E60 e E63.

3.6.2 QP2 - Quais são os principais *softwares* utilizados para mediar este processo?

Para responder a QP2 procurou-se listar todas as ferramentas de *software* cuja utilização foi explorada para facilitar o ensino e aprendizagem de programação de computadores nos estudos pesquisados. Desse modo, constatou-se que 81% dos trabalhos utilizaram algum tipo de ferramenta para auxiliar no ensino de programação, sendo que 21 ferramentas de software foram efetivamente exploradas em atividade prática (Tabela 7), outras 73 ferramentas foram apenas citadas – sem o relato acerca do seu uso efetivo.

Tabela 7 - Ferramentas de *software* efetivamente exploradas

software	Nº publicações	%	Estudos
Scratch	17	25,8%	E2, E3, E4, E7, E8, E9, E11, E15, E24, E28, E29, E47, E50, E60, E61, E63, E65
Alice	16	24,2%	E38, E39, E40, E41, E42, E43, E44, E45, E46, E48, E49, E50, E51, E52, E53, E65
LEGO® Mindstorms	6	9,1%	E6, E10, E56, E57, E60, E65
MIT AppInventor	2	3,0%	E17 e E65
Arduino IDE	2	3,0%	E16 e E33
Blockly	2	3,0%	E27 e E54
Scratch for Arduino	2	3,0%	E33 e E55
CFácil	2	3,0%	E35 e E64
Moodle	2	3,0%	E37 e E44
Pygame	1	1,5%	E12
GameMaker Studio	1	1,5%	E13
iVprog	1	1,5%	E25
FARMA-ALG	1	1,5%	E62
RoboEduc	1	1,5%	E14
RaspiBlocos	1	1,5%	E18
Lord of Cod	1	1,5%	E19
CodeBench (Juíz online)	1	1,5%	E20
RHODES 2.0	1	1,5%	E21
Robocode	1	1,5%	E32
Mblock	1	1,5%	E33
Hour of Code	1	1,5%	E54

Fonte: do autor.

Nesse caso é importante observar que duas ferramentas obtiveram destaque nos trabalhos pesquisados, Scratch (25,8%) e Alice (24,2%), todavia, vale ressaltar que dos estudos que citam Alice, 14 dos 16 foram selecionados a partir do evento

Alice Brasil, cuja ênfase é dada a praticas que abordam o uso de dessa ferramenta para o ensino de programação.

3.6.3 QP3 - Quais benefícios e limitações estão sendo relatados pelo uso das abordagens propostas para o ensino de POO?

Apenas 12% dos estudos analisados exploraram o ensino de POO em sua abordagem. Esse percentual equivale, em números exatos, a 9 (nove) trabalhos, sendo eles: E1, E19, E32, E39, E42, E43, E44, E48 e E51, cujo detalhamento está representado pela Tabela 8.

Tabela 8 - Detalhamento dos estudos que abordam POO

Critério extração		**Qtd**	**Estudo**
É uma RSL		1	E1
Foi testado em sala de aula		6	E19, E39, E43, E44, E48 e E51
Nível de ensino	Ensino Fundamental	1	E43
	Ensino Médio	1	E32
	Ensino Superior	5	E19, E32, E39. E44 e E51
software utilizado	Alice	6	E39, E42, E43, E44, E48 e E51
	Lord of Code	1	E19
	Robocode	1	E32
Estratégia/Método	Avaliação de ferramenta	1	E19
	Jogos	2	E19 e E32
	Mapas conceituais	1	E39
Metodologia	Aprendizagem baseada em problemas	1	E39 e E44
	Aprendizagem por projetos	4	E43, E44, E48 e E51
	Aprendizagem significativa	1	E39

Fonte: do autor.

A partir desse detalhamento é possível que tenhamos uma visão holística dos trabalhos que tratam mais diretamente o ensino de POO. Constatou-se que a grande maioria dos trabalhos foram testados em sala de aula e que a maior parte desses estão direcionados para o ensino superior. Alice foi a ferramenta de *software* mais utilizada e a *Aprendizagem por Projetos* a metodologia mais frequente, seguida de Jogos *e Aprendizagem Baseada em Problemas*. Dos nove trabalhos da lista, apenas o estudo E39 faz referência a uma abordagem pedagógica – *Aprendizagem Significativa*.

Para que possamos responder a QP3 faz-se necessário, para além da análise quantitativa, que se realize a exploração qualitativa desses estudos.

Nesse caso, Souza, Batista e Barbosa (2016) [E1] apontam, em síntese, que as principais dificuldades dos estudantes são:

a) aprender os conceitos de programação;
b) dificuldade dos alunos na aplicação desses conceitos durante a construção de programas;
c) a falta de motivação entre os alunos na realização da atividade de programação.

Para amenizar tais problemas e dificuldades, foram identificadas possíveis soluções:

a) utilização de visualização de programas e algoritmos;
b) utilização de *serious* games;
c) desenvolvimento de ambientes pedagógicos para o ensino e aprendizagem de programação.

Souza et al. (2016) no estudo E19, comentam sobre a dificuldade dos estudantes em abstrair a aprendizagem de programação, pois é um processo complexo que envolve habilidades diversas exigindo alta carga cognitiva para que os estudantes obtenham sucesso. Os autores destacam ainda que nos cursos de computação, as disciplinas que envolvem programação de computadores são as que apresentam maiores índices de desistência ou reprovação. Nesse trabalho, os autores propõem o desenvolvimento de ambiente lúdico (*Lord of Code*[13]) baseado na utilização de jogos para estimular a aprendizagem de programação. Nesse caso, uma avaliação piloto foi realizada com alunos de graduação em Licenciatura em Computação do IFMG – campus Ouro Preto, os quais depois do teste avaliaram a ferramenta utilizada, classificando o nível de auxílio da ferramenta entre “bom” e “poderia ser melhor”. Os autores reconhecem a necessidade de se realizar testes mais elaborados para se obter dados mais concretos acerca da utilização dessa ferramenta.

Semelhante a essa experiência, Amaral, Braga e Silva e Pantaleão (2015)[E32] utilizaram a ferramenta Robocode[14] para organizar uma olimpíada de programação envolvendo desafios gamificados em que participaram alunos do curso de graduação em Engenharia Eletrônica e de Telecomunicações da Universidade Federal de Uberlândia (UFU) em um projeto de extensão que envolveu alunos do ensino Médio de escolas públicas da cidade de Patos de Minas/MG. Essa experiência consistiu na organização de um campeonato de programação em que os participantes deveriam

[13] Ambiente em desenvolvimento – (http://www.br-ie.org/pub/index.php/sbie/article/view/6820/4705)
[14] Link: http://www.robocodebrasil.com.br/

programar um robô incluindo a ele funcionalidades para combater em um cenário de guerra. Quanto mais aprimoradas fossem as funcionalidades desenvolvidas para esse ator, maiores seriam as chances de sucesso na competição. Nesse caso, o fator determinante da atividade envolve agilidade, pensamento computacional e conhecimento em linguagem de programação – que, no caso da ferramenta, é orientada a objetos.

> Com o desenvolvimento desse trabalho, percebeu-se, na pratica, o quanto o uso de jogos educacionais tem se mostrado uma estratégia de ensino promissora. Acredita-se que a aprendizagem por meio da investigação venha a desenvolver o senso crítico dos alunos envolvidos, estimulando a busca pelo conhecimento e a competição. Com o Robocode, os alunos do Ensino Médio tiveram a oportunidade de desenvolver seu raciocínio logico e aplicar conceitos básicos de Matemática em um ambiente lúdico de ensino, atrativo e interativo (AMARAL, BRAGA E SILVA & PANTALEÃO, 2015, p.206).

Outro caso de sucesso do uso de ferramentas para o ensino de POO é relatado por Antonio e Ferro (2011) [E39], o qual aborda a utilização de: Ambiente virtual de aprendizagem (AVA) - Moodle; técnica de PLB; ambiente de programação Alice 3D e Mapas conceituais. Tais técnicas e ferramentas foram utilizadas em conjunto para facilitar a aprendizagem de POO, conforme abordagem proposta por Antonio e Ferro (2010). Assim, o AVA foi utilizado para servir de canal de comunicação/interação entre professores e alunos. A técnica PLB foi proposta para incentivar que os acadêmicos fossem atores ativos do processo de aprendizagem. O ambiente Alice 3D foi utilizado como ferramenta de desenvolvimento e os Mapas Conceituais para apoiar a elicitação de requisitos na fase de engenharia de *software*.

A experiência relatada por Antonio e Ferro (2011) apresentou bons resultados no que tange ao de ensino de POO, visto que a metodologia consistiu na aplicação de um questionário antes do desenvolvimento da dinâmica e depois de concluído o processo. Nesse caso, os autores constataram que “a aplicação da ferramenta Alice apoiada pelo ambiente virtual mostra claramente expressiva melhoria do conhecimento de tais conceitos de orientação a objetos, depois da aplicação da abordagem apresentada nesse trabalho” (ANTONIO e FERRO, 2010). Os autores destacam ainda a importância do *software* utilizado, enfatizando que “o uso do *software* Alice, despertou o interesse e facilitou o entendimento desses conceitos”.

Craveiro et al. (2011), no E42 destacam as funcionalidades do *software* Alice para o ensino de POO, pois consiste em um ambiente lúdico de desenvolvimento no qual é possível trabalhar a modelagem de classes de maneira diferenciada do que

normalmente se trabalha nas escolas, pois, em cada ator do cenário (objeto), é possível a configuração de características (atributos) e ações (métodos) próprias. Ademais, cada objeto pertence a uma classe e sua utilização depende de reutilização da classe mãe (herança). Ainda, cada classe possui métodos pré-definidos que podem ser simplesmente utilizados ou personalizados (polimorfismo, sobrecarga e sobrescrita).

A exemplo de outros trabalhos, o Ferreira (2012) utiliza o *software* Alice para promover um festival de animação para estudantes das séries finais do Ensino Fundamental. O referido festival foi chamado de Animalice inspirado no festival "Anima Mundi[15]" e consiste em uma competição que envolve a disciplina de Robótica, Programação e temas interdisciplinares. Segundo o autor desse trabalho:

> A Robótica é uma disciplina nova, multidisciplinar que possibilita o acesso a novas tecnologias. A utilização de ferramentas que faça uso da programação se insere muito facilmente no seu contexto. Desta forma, o *software* Alice surge como um instrumento para a aprendizagem de programação altamente instigante devido ao seu caráter lúdico, que permite abordar temáticas interdisciplinares, o trabalho em equipe e a criatividade para desenvolver animações e jogos. (FERREIRA, 2012)

Antonio e Ferro (2013) apresentaram no E44 a utilização do *software* Alice no ensino de programação para estudantes de Sistemas de Informação. Para esse trabalho, os autores propuseram o desenvolvimento de atividades predefinidas organizadas na forma de problemas a serem solucionados. A dinâmica consiste na realização da sequência de cinco atividades em que uma é pré-requisito da próxima. Ao final da experiência, os autores destacam que os resultados são:

> [...] promissores na tentativa de inserir o *software* Alice como parte das atividades didáticas pedagógicas para o ensino de programação. Os resultados encontrados, especialmente do ponto de vista da avaliação qualitativa da aplicação das atividades ao longo do semestre, permitiram evidenciar a motivação dos alunos ao elaborarem as atividades sugerindo a real viabilidade de aplicação do *software* no contexto educacional do curso de Sistemas de Informação (ANTONIO E FERRO, 2013).

Por fim, no E51, Andrade, Moreno e Rossi (2016) seguem nessa linha ao propor o desenvolvimento de uma atividade que, através da construção de um cenário no software Alice, permite que o estudante explore os conceitos básicos de orientação a

[15] O Festival Internacional de Animação do Brasil, ou Anima Mundi, é uma organização de animações que ocorre anualmente no mês de julho nas cidades do Rio de Janeiro e São Paulo, no Brasil.

objetos de maneira lúdica. Essa atividade foi desenvolvida para estudantes do ensino superior que estão cursando a disciplina de Análise, Projeto e Desenvolvimento de Sistemas. No relato da experiência, as autoras não deixam claro, na metodologia utilizada, detalhes relacionados à amostra do público. Todavia, é citado no texto, em síntese, que a partir da "percepção qualitativa" das autoras os resultados são os seguintes:

> • A atividade proposta com o apoio do software ALICE, permite a aplicação de técnicas ativas de aprendizagem, onde o aluno vivencia e adquire o conhecimento de maneira proativa e lúdica;
>
> • O aluno vivencia a elaboração de projeto, para da solução;
>
> • O aluno aplica os conceitos básicos de orientação a objetos de uma maneira implícita na criação da cena projetada; (ANDRADE, MORENO e ROSSI, 2016)

3.7 CONSIDERAÇÕES DA RSL

Essa RSL pesquisou publicações que abordassem metodologias de ensino de programação apoiadas pelo uso de tecnologias. Foram incluídos trabalhos publicados no período de 1º de janeiro de 2010 a 31 de dezembro de 2016, nos principais periódicos e anais de eventos da área de Informática na Educação no Brasil: SBIE, WIE, RBIE, WAlgProg, RENOTE e Alice Brasil. Foram selecionadas 166 publicações dentre as quais 66 foram incluídas para a extração dos dados.

Constatou-se que 60% dos estudos incluídos são dos últimos dois anos do período selecionado, o que demonstra que há um movimento crescente em relação a pesquisa de praticas de ensino de programação. No que tange ao nível de ensino dos trabalhos, percebe-se que a grande maioria se concentra no ensino médio e superior (42% e 38%, respectivamente). Todavia, pode-se identificar a existência de alguns ensaios de ensino de programação para o Ensino fundamental, o qual tem o intuito de incentivar o gosto pela área da computação com possibilidade de opção para essa carreira profissional.

Nos estudos incluídos foram exploradas 21 ferramentas de apoio ao ensino de programação, com destaque para ALICE e SCRATCH, os quais juntos estiveram presentes em 50% destes trabalhos. Quanto ao nível de programação abordada, os quantitativos encontrados são os seguintes: 85% dos estudos fazem referência a "Algoritmos ou Lógica de programação"; 32% a "Programação estruturada" e; 12%

“Programação Orientada a Objetos”. Cabe salientar que vários trabalhos utilizaram abordagens em mais de um nível de programação.

No que tange a metodologias utilizadas, verificou-se que as metodologias “*Aprendizagem por projetos*” e “*Aprendizagem baseada em problemas*”, bem como a utilização de jogos e *robótica* receberam destaque nos trabalhos pesquisados, pois juntas, estiveram presentes em 45% dos estudos incluídos.

Embora seja verificada grande quantidade de estudos relacionados a práticas de ensino de programação, fica evidente a escassez de trabalhos que abordem especificamente o ensino de POO em relação aos demais, o que caracteriza a existência de um espaço a ser melhor explorado pela produção de referencial e métodos que auxiliem tais práticas.

O mesmo acontece em relação às abordagens pedagógicas, as quais são pouco citadas nos trabalhos, sendo que *Aprendizagem significativa*, *Construtivismo* e *Construicionismo* são as únicas abordagens relatadas nos estudos incluídos. Embora a utilização de jogos e robótica sugiram, respectivamente, a utilização do Construcionismo e Construtivismo, nem sempre utilização desses referenciais pedagógicos são explicitamente citados nos trabalhos pesquisados.

A RSL buscou realizar uma prospecção que pudesse apresentar um “estado da arte” acerca das principais ferramentas e metodologias utilizadas no ensino da programação de computadores, em especial àquelas que abordem a POO. Nesse sentido, várias metodologias obtiveram destaque, porém, no caso específico de POO, a PBL, Aprendizagem por Projetos e Jogos receberam maior atenção. De todo modo, na maioria dos casos, tais metodologias eram utilizadas em conjunto com outras dinâmicas, inclusive Jogos. Tal constatação ratifica a intencionalidade inicial do projeto global em abordar jogos para o ensino de POO.

Todavia, faz-se necessário optar por uma das ferramentas em destaque para utilizar no contexto de “ensinar e aprender POO pelo desenvolvimento de jogos” que é objetivo desse trabalho. Nesse caso, ao analisar a “radiografia” do ensino de POO detalhado na seção 3.6.3, observa-se que foram utilizadas apenas três ferramentas para essa finalidade: Alice, Lord of Code e Robocode. Para tanto, partiu-se para uma análise prática de cada ferramenta, a fim de se obter sensações que possibilitem escolher a que melhor se adapta a intencionalidade do projeto.

Ao analisar Alice, constatou-se que possui grande potencial para o ensino de POO, visto que desenvolve toda a dinâmica sob a perspectiva de orientação a objetos,

é uma ferramenta gratuita e fácil de utilizar. Contudo, embora possa criar algum tipo de jogo, Alice foi projetado para a construção de cenários de mundo virtuais, com objetivo de criar pequenos clipes interativos, programáveis que utilizam blocos para desenvolvê-los. Nesse caso, a utilização de Alice nessa proposta tende a não contemplar os objetivos.

Lord Of Code, a segunda ferramenta analisada, consiste de um jogo em desenvolvimento que apresenta um algoritmo defeituoso e solicita ao jogador uma resposta adequada à sua solução – opera com sistema de pontuação e, por se tratar de um jogo em desenvolvimento não foi possível testá-la na prática. Nesse caso, essa ferramenta não corresponde à necessidade do projeto.

Já a ferramenta Robocode, consiste na programação de um robô que é um tanque de guerra. Quanto mais aprimorada a programação, melhor será o desempenho do robô no “combate”. Todavia, as classes de base do robô já contam com métodos pré-definidos, cabendo ao programador utilizá-lo de maneira conveniente. Existe a possibilidade de criar métodos de autoria, porém tal programação demanda carga cognitiva considerável para se obter sucesso. É uma ótima ferramenta para ensinar POO, porém exige domínio considerável de programação e os jogos nele desenvolvidos podem abordar apenas a temática de guerra.

A fim de se esgotar todas as possibilidades na escolha de uma ferramenta adequada verificaram-se também possibilidades entre os itens que estiveram elencados fora do contexto de “ensinar e aprender POO pelo desenvolvimento de jogos”, incluindo-se as ferramentas que foram apenas citadas – com o critério de que utilizassem o desenvolvimento de jogos como fundamento básico. Nesse caso, selecionou-se para análise as seguintes possibilidades: Kodu Game Lab, Greenfoot, Scratch, GameMaker Studio e Construct2.

Ao analisar na prática cada uma dessas ferramentas, percebeu-se que Kodu Game Lab foge da proposta, pois utiliza programação visual com poucas possibilidades de ensinar os conceitos elementares de POO, pois não há interação à nível de código. O mesmo acontece com Game Maker Studio e Construct2, nas quais há interação com código, porém são dedicadas a criação rápida de jogos, contendo

assets[16] predefinidos com pouca intervenção do programador. O Scratch possui condições de desenvolver jogos, porém sua programação em blocos oferece boas possibilidades para o ensino de programação em fase inicial – algoritmos e programação estruturada. Por outro lado, sua utilização já é amplamente consolidada no meio acadêmico.

Por fim, o Greenfoot mostrou-se bastante adequado para essa proposta, pois aborda o desenvolvimento de jogos em sua essência e, utiliza exclusivamente a POO em sua estrutura de desenvolvimento. Trata-se de um ambiente desenvolvido exclusivamente para ensino de POO, contém bons referenciais de apoio e sua utilização é gratuita. O tipo de jogo a ser desenvolvido depende da criatividade do desenvolvedor e possui uma interface de simples utilização.

[16] Fragmentos de software pré-definidos que podem ser agregados na forma de complemento em uma ferramenta de desenvolvimento de jogos. Em geral são criados e comercializados em lojas especializadas.

4 ASPECTOS METODOLÓGICOS

Este capítulo tem o intuito de descrever os materiais e métodos que foram utilizados para o desenvolvimento da pesquisa. Nesse sentido, as seções que seguem visam elencar os detalhes da proposta, bem como o explicitar o referencial metodológico utilizado. Nesse caso, é fundamental esclarecer que a execução do trabalho científico perpassa, para além do rigor metodológico utilizado, pela compreensão do espaço em que ele foi desenvolvido, o que envolve o reconhecimento do público, materiais e métodos. Com relação ao método de pesquisa, Matias-Pereira (2012, p.34) ressalta que:

> O método pode ser entendido como o roteiro, procedimentos e técnicas utilizados para se alcançar um fim ou pelo qual se atinge um objetivo. O método científico é o conjunto de procedimentos e técnicas utilizados de forma regular, passível de ser repetido, para alcançar um objetivo material ou conceitual e compreender o processo de investigação.

Nesse caso, esse capítulo está dividido em sete seções a fim de descrever todos os passos da pesquisa a fim de se alcançar os seus objetivos.

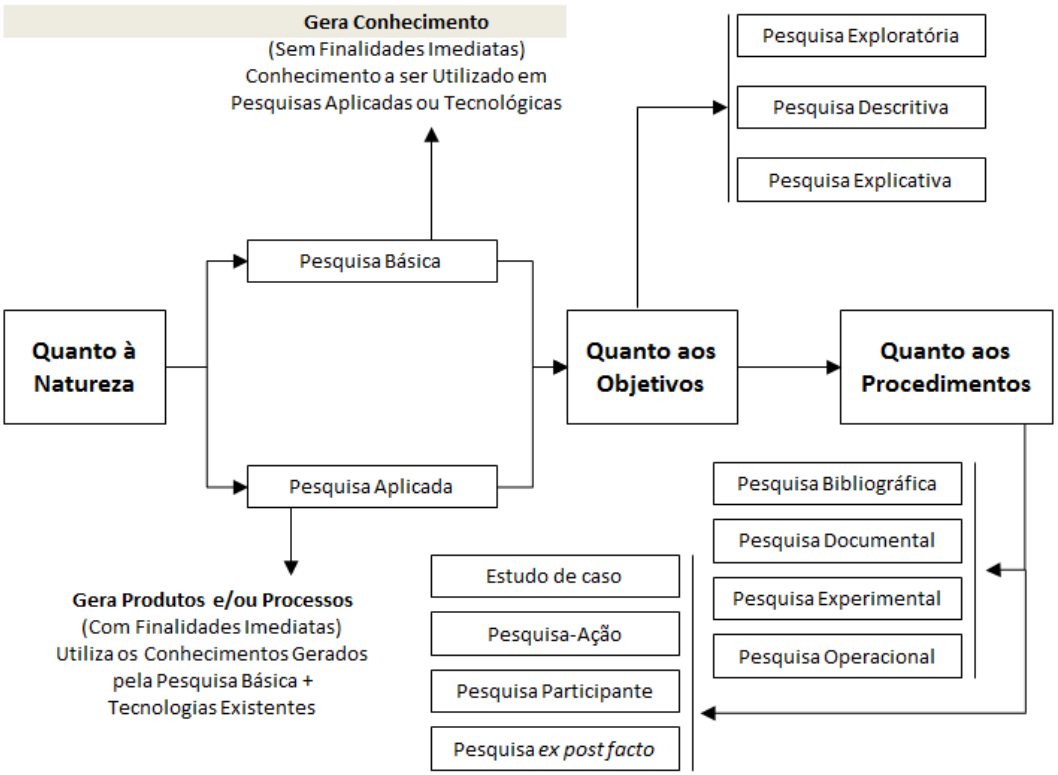
4.1 CLASSIFICAÇÃO METODOLÓGICA

Para que se possa organizar o processo de pesquisa, escolher os procedimentos metodológicos é fundamental, pois, dessa maneira, os objetivos definidos para a pesquisa serão mais facilmente alcançados e, consequentemente, o pesquisador conseguirá solucionar o problema que é objeto da investigação. Nesse caso, para facilitar a abstração do método escolhido é importante classificar a pesquisa de acordo com a problemática e os objetivos. Sobre essa temática Gil (2010) enfatiza que é próprio da racionalidade humana classificar fatos e acontecimentos. O autor destaca também que essa atividade serve para organizar o pensamento e facilitar o entendimento de situações e problemas. O referido autor destaca que é possível classificar a pesquisa quanto à *área de conhecimento*, sua *finalidade* e os *métodos empregados*.

De outro lado, Matias-Pereira (2012) utiliza um formato mais clássico de classificação que, em essência, segue estrutura semelhante à proposta de Gil (2010). Nesse caso, Matias-Pereira (2012, p.86-87) enfatiza que existem várias formas de se classificar as pesquisas, porém, o autor sugere a seguinte estrutura: *Quanto à*

natureza; Quanto à abordagem do problema; *Quanto aos objetivos* e *Quanto aos procedimentos técnicos* (ilustrado pela Figura 7). Assim, para classificar a investigação em tela serão utilizados referenciais desses dois autores, pois, em conjunto, apresentam rico manancial teórico acerca do método científico atualmente utilizado.

Figura 7 - Natureza, Objetivos e Procedimentos da Pesquisa Científica.



Fonte: Matias-Pereira (2012).

Quanto à abordagem do problema, Matias-Pereira (2012) define que podemos classificar uma pesquisa sob a ótica *quantitativa* ou *qualitativa*. Assim, para alcançar uma solução para o problema objeto dessa investigação, faz-se necessário analisar qualitativamente os comportamentos e sensações dos sujeitos envolvidos no processo, cujas variantes não podem ser previstas ou mensuradas apenas em números. Nesse caso, a investigação deve ser abordada sob a perspectiva de *pesquisa qualitativa*. Por outro lado, será necessário – como veremos mais adiante - valer-se de técnicas de tabulação de dados, resultados de procedimentos técnicos a serem adotados. Por este motivo, a pesquisa toma os rumos de uma abordagem *quantitativa*. Claramente verifica-se que não se pode classificar a abordagem para essa investigação apenas como "*quanti*" ou "*quali*", pois para o desenvolvimento da

proposta será necessário utilizar técnicas características de ambas as abordagens. Sobre essa situação, Matias-Pereira (2012, p.87-88) ressalta que:

> Entretanto, em que pesem as diferenças nessas abordagens, elas não podem ser vistas como excludentes. No mundo contemporâneo fica cada vez mais evidente que as duas abordagens – quantitativa e qualitativa – são complementares, ou seja, mesmo com problemas e tópicos diferentes, mas igualmente importantes.
>
> É oportuno ressaltar que a literatura recente mostra que o emprego dessas duas abordagens na pesquisa de um mesmo problema, em geral, tende a apresentar um resultado mais consistente.

Assim, por abranger questões relacionadas à amostragem de resultados a partir de coleta de dados – elementos próprios da pesquisa quantitativa – e na descrição e análise dos dados coletados, configurando a pesquisa qualitativa. A classificação coerente para essa investigação define-se como *pesquisa quanti-qualitativa*.

Em relação à natureza da pesquisa, a presente pesquisa detém características de pesquisa *aplicada*, pois objetiva gerar conhecimentos que resultarão em uma aplicação prática visando encontrar a solução de problemas específicos (Matias-Pereira, 2012). No que se refere aos objetivos, essa pesquisa tem o propósito de explorar/dinamizar o ensino de programação orientada a objetos, buscando melhorar técnicas e práticas pedagógicas. Nesse caso, pode-se determinar que os seus objetivos se enquadram à *pesquisa exploratória,* pois visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou formular hipóteses de melhoria no ensino (GIL, 2010; MATIAS-PEREIRA, 2012).

Por se tratar de um método que envolve interação entre pesquisador e pessoas implicadas na situação investigada; por ter caráter de esclarecer o problema; e por acompanhamento constante das ações este estudo trata-se de uma *pesquisa-ação* (THIOLLENT, 2011). A escolha desse procedimento também se deve pelo fato de aceitar os diferentes papéis desempenhados pelo pesquisador, que de um momento é de pesquisado e de outro é pesquisador cujo objeto de reflexão é a sua própria prática (BARBIER, 2007).

4.2 AMOSTRA/POPULAÇÃO ALVO

O ambiente da pesquisa é o Instituto Federal Farroupilha *campus* Santo Augusto/RS, que possui quatro cursos Técnicos Integrados ao Ensino Médio: Alimentos, Agropecuária, Administração e Informática e quatro cursos superiores:

Tecnólogo em Alimentos, Tecnólogo em Agronegócio, Licenciatura em Computação e Licenciatura em Ciências Biológicas. Para esse estudo foram selecionadas duas turmas de 3º ano curso do Técnico em Informática Integrado ao Ensino Médio do IFFAR *campus* Santo Augusto, cada uma com aproximadamente entre 20 e 26 estudantes, cuja intervenção aconteceu na disciplina de Programação III (ministrada pelo pesquisador).

4.3 FASES DA PESQUISA

A pesquisa aconteceu em três etapas:

1) Intervenção piloto, na disciplina de Programação III da turma do 3º ano do Curso Técnico em Informática Integrado ao Ensino Médio, no ano letivo de 2017;
2) Segunda intervenção, que aconteceu também na disciplina de Programação III do mesmo curso, porém, com a turma do 3º ano de 2018;
3) Tabulação e análise dos dados coletados durante as intervenções.

Cada intervenção consistiu no desenvolvimento de Unidades de Estudo, que têm como objetivo ensinar, com o uso do Greenfoot, os conceitos de orientação a objetos relacionados à Classe, Objeto, Atributos, Métodos, Instanciação, Herança e Polimorfismo.

Unidades de Estudo, segundo Filatro e Cairo (2015), fazem referência ao desenvolvimento de um tipo de Metamodelo Educacional, cuidadosamente elaborado para que atenda às necessidades de aprendizagens dos estudantes. No caso desta pesquisa, as Unidades de Estudo envolvem desenvolvimento de jogos (com ênfase nos conceitos de POO) e questionários. Sendo que, cada uma das etapas está detalhada na seção 4.5.3.

O embasamento pedagógico de toda a proposta foi norteado pelo *Ciclo de Aprendizagem de Kolb* (detalhado na seção 4.6), pois contempla em sua base teórica versatilidade semelhante à dinâmica proposta pelos jogos, proporcionando condições de aprendizagem a partir de um ciclo contínuo entre agir, refletir, conceitualizar e aplicar.

4.4 ORGANIZAÇÃO

Esta seção tem como objetivo apresentar a organização das intervenções realizadas com as turmas selecionadas. Nesse sentido, cada umas das intervenções foram planejadas para obedecer, de maneira sintética, os passos ilustrados na Figura 8.

Figura 8 - Passos realizados em cada uma das intervenções

Fonte: do autor.

4.4.1 Intervenção piloto

A intervenção piloto realizou-se nos meses de novembro a dezembro de 2017, com estudantes do 3º ano do Curso Técnico em Informática Integrado ao Ensino Médio 2017 e consistiu na seguinte dinâmica, executada sequencialmente:

a) Professor apresentou/retomou os conceitos de POO aos estudantes envolvidos;
b) Cada estudante criou um Mapa Conceitual acerca dos conceitos de POO apresentados;
c) Estudantes realizaram as atividades propostas nas três Unidades de Estudo (Apêndices A, B e C), as quais contemplam o desenvolvimento de jogos utilizando Greenfoot e questionamentos específicos de POO;
d) Estudantes refizeram seu Mapa Conceitual sobre os conceitos de POO;

e) Estudantes responderam ao questionário de opinião sobre o uso do Greenfoot como ferramenta auxiliar no ensino de POO (APÊNDICE G);
f) Pesquisador realizou avaliação quantitativa do progresso do mapa e analisou qualitativamente os dados coletados;

4.4.2 Segunda Intervenção

A segunda intervenção realizou-se entre os meses de março a junho de 2018, iniciando com uma análise da intervenção piloto, a fim de realizar ajustes necessários nos materiais utilizados.

A sequência consistiu em ampliar o rol de Unidades de estudo, a fim de contemplar demais conceitos de POO e aplicar a mesma dinâmica (Figura 8) com os estudantes do 3º ano do Curso Técnico em Informática - turma de 2018. Nesse caso, o procedimento foi o mesmo realizado com os estudantes da turma de 2017, exceto pelas unidades de estudo trabalhadas, pois no caso desta turma, para que não se tornasse um processo muito cansativo, foram realizadas apenas as atividades das unidades de estudo 2.1 e 2.3 (Apêndices D e F).

4.4.3 Terceira etapa

A última fase desse projeto constituiu o momento final da práxis pedagógica aonde os dados foram analisados quanti e qualitativamente e os materiais produzidos foram ajustados para que possam ser reutilizados na forma de itinerários pedagógicos para o ensino de POO. Salienta-se que a análise quantitativa é compreendida pela tabulação e análise dos dados dos questionários e das notas obtidas pelos estudantes na produção dos mapas conceituais em POO. Ainda, em relação às notas dos mapas conceituais, salienta-se que foram submetidas à teste estatístico paramétrico - Teste t de *student* (teste t)[17] pareado (MOORE, 2005) a fim de verificar possíveis progressos dos alunos depois de realizar as atividades.

[17] Um teste t poderá ser usado para testar a diferença entre duas médias populacionais quando uma amostra for selecionada aleatoriamente de cada população. Para que se possa realiza-lo, cada população deve ser normal e cada membro da primeira amostra precisa estar associado a um membro da segunda amostra (LARSON, 2007, p.316).

4.5 MATERIAIS UTILIZADOS

Esta seção tem por objetivo apresentar os materiais utilizados no desenvolvimento da proposta desta pesquisa. Nesse caso, na sequência, são detalhados os seguintes recursos:

a) Greenfoot;
b) Mapas conceituais;
c) Unidades de estudo;
d) Ciclo de Kolb.

Tais materiais auxiliaram no decurso da pesquisa, organizado conforme esquema ilustrado pela Figura 9.

Figura 9 - Materiais utilizados no desenvolvimento da pesquisa

Fonte: do autor.

Os materiais utilizados na pesquisa serão detalhados nas próximas seções, a fim de proporcionar maior compreensão da sua contribuição no processo.

4.5.1 Greenfoot

O Greenfoot é uma IDE[18] para criação de jogos em 2D. Foi desenvolvida por pesquisadores das universidades de Kent, na Inglaterra, e a Universidade de Deakin, Melbourne, Austrália no ano de 2006. Seus idealizadores são Michael Kölling e Poul Henriksen. A ferramenta utiliza como linguagem de programação o JAVA, possibilitando à estudantes de computação uma perspectiva mais lúdica acerca da utilização do paradigma de POO por meio do Greenfoot.

Em essência, Greenfoot, foi desenvolvido para fins educacionais, sendo que o seu objetivo principal é proporcionar um ambiente visual capaz de facilitar o aprendizado de programação orientada a objetos a partir da construção de cenários de jogos. A tradução literal do nome da ferramenta é “Pé Verde”, o que gera certa curiosidade acerca dessa escolha. Nesse caso, segundo um de seus criadores Michael Kölling, o nome Greenfoot foi escolhido a partir de uma antiga fábula dos aborígenes australianos, os quais acreditavam na existência um ser mitológico cujo nome era Greenfoot, e tinha a responsabilidade de iluminar com conhecimento os integrantes da tribo. Assim, a logomarca padrão da ferramenta é o desenho de um pé verde (Figura 10).

Figura 10 - Logomarca Greenfoot

Fonte: Greenroon (2017).

Trata-se de software de desenvolvimento com bons recursos de programação, sendo que o código fonte da sua IDE encontra-se sob licença GNU General Public Licence[19] versão 2, com exceção do classpath[20]. Desse modo, no site oficial da

[18] IDE - do inglês Integrated Development Environment ou Ambiente de Desenvolvimento Integrado, é uma ferramenta computacional criada com o propósito de agilizar processos de desenvolvimento de software, pois integra diversos componentes relacionados a essa atividade.

[19] GNU General Public Licence - é a designação da licença de software idealizada por Richard Matthew Stallman de acordo com as definições de software livre.

[20] Classpath - É uma variável de ambiente utilizada pela linguagem JAVA para armazenar informações acerca das localizações dos arquivos e bibliotecas necessários para a compilação e execução de um

ferramenta (www.greenfoot.org) é possível realizar o *download* do Greenfoot e do seu código fonte. No site, também estão disponíveis tutoriais, suporte, cenários e exemplos e conta ainda com uma comunidade de usuários onde estudantes e professores podem se filiar para trocas de experiências e ajuda cooperativa. Assim, a utilização da ferramenta é facilitada por conta da grande quantidade de informações disponíveis e exemplos de utilização.

Com o Greenfoot é possível a criação de jogos em cenários 2D de maneira fácil e intuitiva. Por utilizar o Java, é uma ferramenta totalmente orientada a objetos, contendo, atualmente, um rol de nove classes nativas para facilitar o processo de criação:

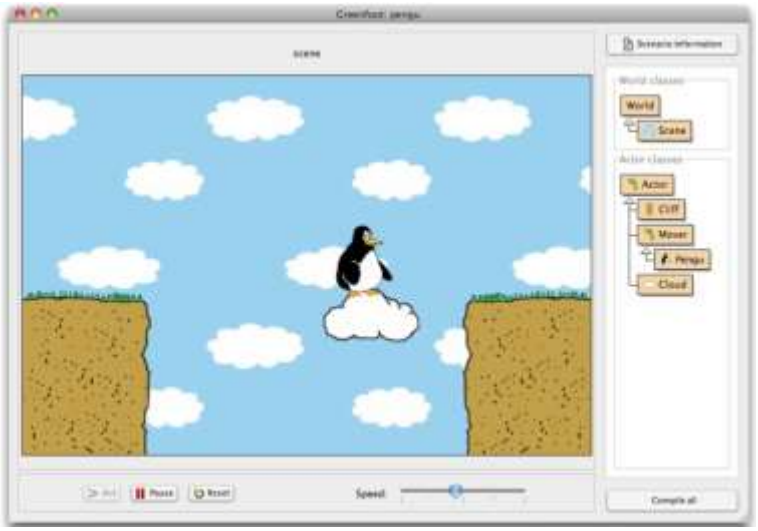
a) ***Actor*** - Contém os métodos disponíveis para os atores do jogo.
b) ***Color*** – Usada para preencher cores na tela.
c) ***Font*** - A Fonte pode ser usada para escrever texto na tela;
d) ***Greenfoot*** – Usado para se comunicar com o Ambiente Greenfoot em si;
e) ***GreenfootImage*** – contém métodos para apresentação e manipulação de imagens.
f) ***GreenfootSound*** - contém métodos para apresentação e manipulação de sons;
g) ***MouseInfo*** – Métodos para capturar eventos do Mouse;
h) ***UserInfo*** - A classe UserInfo pode ser usada para armazenar dados permanentemente em um servidor e para compartilhar esses dados entre diferentes usuários, quando o cenário é executado no site Greenfoot;
i) ***World*** – Responsável por implementar os métodos relacionados ao “mundo” ou cenário do jogo.

Como é premissa no desenvolvimento de jogos, tudo acontece em um “Mundo” ou cenário onde o jogo acontece. Nesse caso, a primeira classe a ser personalizada e instanciada deve ser a classe “World” que, como já comentado, é uma das classes nativas da ferramenta. Depois de criar um “Mundo”, a ele são incorporados atores que, em essência, são os objetos que irão interagir com o jogador e entre eles próprios para que o enredo do jogo se desenrole. É importante salientar que todos os objetos colocados sobre o mundo são atores – independentemente de serem animados ou inanimados. A Figura 11 ilustra a tela principal do Greenfoot.

---

programa criado nessa linguagem. Dessa forma, essa variável desempenha papel fundamental no funcionamento de qualquer software desenvolvido em JAVA.

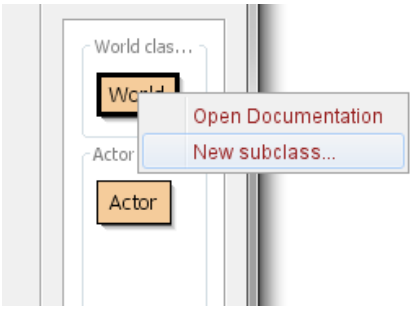
Figura 11 - Tela principal do Greenfoot



Fonte: Kölling (2010).

Um fato importante em relação às classes “World” e “Actor” é que ambas são abstratas, nesse caso, não é possível instanciar objetos a partir delas, pois servem apenas de modelos para subclasses. Assim, para a sua utilização, é obrigatória a criação de subclasses. A tarefa de criar subclasses no Greenfoot é facilitada pela IDE e pode ser efetivada apenas com o uso do mouse (Figura 12).

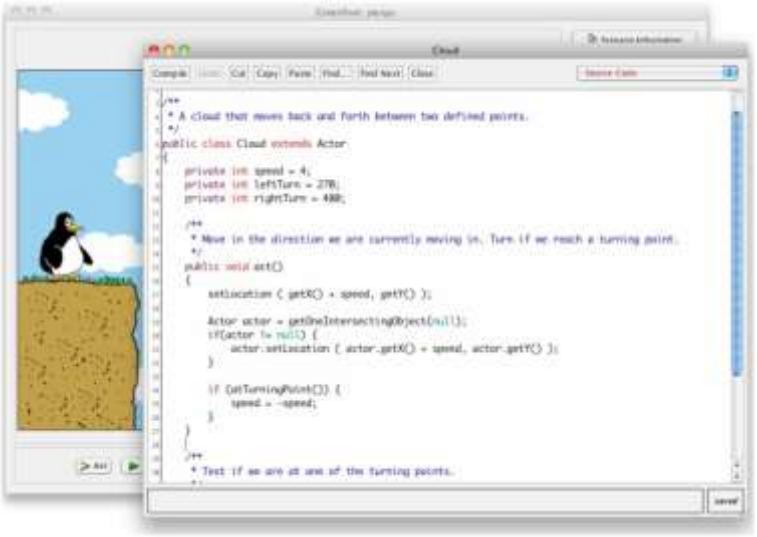
Figura 12 - Criando subclasse de World



Fonte: do autor.

Para além da tela principal é possível acessar o código fonte das classes criadas (Figura 13), com possibilidade de personalização dos códigos, criação de outros construtores, métodos e a sobrecarga e reescrita de métodos ancestrais – tudo em consonância com a proposta do paradigma de POO. Usuário do Greenfoot podem ainda criar e incorporar classes personalizadas a fim de obter melhores resultados.

Figura 13 - Editor de códigos do Greenfoot



Fonte: Kölling (2010).

Segundo Kölling (2010) a utilização da ferramenta está baseada na possibilidade de criação e utilização de vários objetos da mesma classe em um mesmo mundo, sendo que os métodos podem ser executados em cada um deles individualmente. Ainda, o estado de cada ator (objeto) é individual, ilustrando a independência entre eles, pois cada objeto tem valores diferentes dos demais. Da mesma forma, é possível perceber que objetos de uma mesma classe possuem os mesmos campos (atributos) e podem realizar as mesmas ações (métodos).

Essa forma de abordagem e de interações é, pedagogicamente valiosa, pois, permite aos professores apresentarem alguns dos conceitos mais importantes e fundamentais da POO de uma maneira facilmente compreensível (KÖLLING,2010). O autor elenca ainda os conceitos fundamentais abordados na utilização da ferramenta:

a) Um programa consiste em um conjunto de classes;
b) A partir das Classes, podemos criar objetos;
c) Múltiplos objetos podem ser criados a partir de uma classe;
d) Todos os objetos da mesma classe oferecem os mesmos métodos e têm os mesmos campos;
e) Cada objeto possui seus próprios valores para seus campos (cada objeto detém um estado);
f) Interagimos com os objetos chamando seus métodos;
g) Métodos podem ter parâmetros;
h) Métodos podem retornar valores;
i) Parâmetros e valores de retorno têm tipos.

Os itens elencados pelo autor fazem referência a conceitos fundamentais da POO que, tradicionalmente, são difíceis de ensinar e aprender, pois, são muito

abstratos. Nesse sentido, o objetivo do Greenfoot está em tornar concretas essas abstrações, de forma a proporcionar que o estudante visualize explicitamente as interações realizadas no código fonte.

Ainda é possível acrescentar a esse benefício, a ludicidade proporcionada pelos jogos que, naturalmente instiga os estudantes em ir além da proposta da aula, motivados pela curiosidade em conhecer outras funcionalidades e possibilidades de personalização de um jogo de sua autoria.

4.5.2 Os Mapas Conceituais

Mapa conceitual é um recurso para organizar e representar a estrutura cognitiva do indivíduo. Pode ser utilizado para ilustrar de maneira gráfica as relações entre os conceitos de um determinado assunto. Nesse caso, mapas conceituais operam como uma importante ferramenta para que o aprendiz organize seu pensamento e possibilite que o professor avalie a aprendizagem dos seus alunos. Tal recurso foi desenvolvido na década de setenta pelo pesquisador norte-americano Joseph Novak, com base na teoria da aprendizagem significativa de David Ausubel (MOREIRA, 2011).

Para Ausubel, aprendizagem significativa é um processo pelo qual uma nova informação se relaciona com um aspecto relevante da estrutura cognitiva do indivíduo (PEÑA et al., 2005). Assim, pode-se dizer que a aprendizagem é dita significativa quando uma nova informação adquire significado para o aprendiz. Conforme a teoria da aprendizagem significativa, o indivíduo estabelece tais significados quando consegue aderir o conceito que está tentando aprender a conceitos, valores e ideias já existentes em sua estrutura cognitiva.

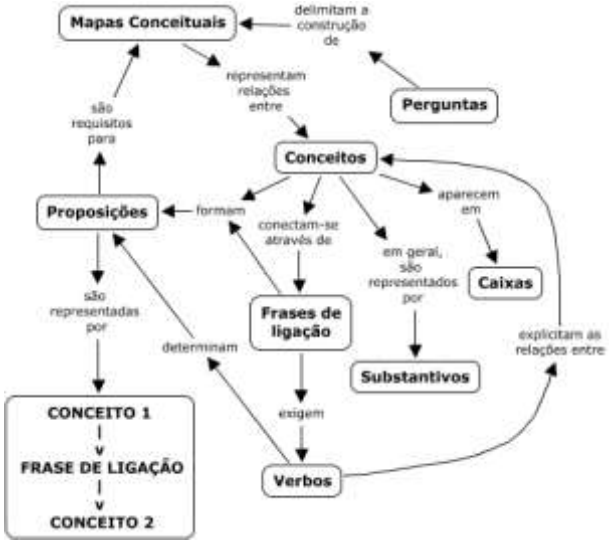
Aprender significativamente implica necessariamente em atribuir valores pessoais ao novo conhecimento, caso contrário, se não houver relação com os conhecimentos preexistentes a aprendizagem é mecânica não significativa. É preciso estabelecer equivalências, agrupando ideias relacionadas à experiência com os conceitos do indivíduo.

A aprendizagem mecânica, segundo Ausubel, consiste na aprendizagem de novas informações com pouca ou nenhuma interação com conceitos relevantes existentes na estrutura cognitiva do indivíduo, ou seja, a nova informação é armazenada de maneira arbitrária, desta forma não estabelece relação com as

estruturas já armazenadas (MOREIRA e MANSINI, 2009). Na verdade, Ausubel reconhece a existência de benefícios da aprendizagem mecânica, pois, ela pode ser necessária quando o aprendiz tem contato com conhecimentos completamente novos. Segundo Moreira e Masini (2009, p.19) “[...] a aprendizagem mecânica é sempre necessária quando um indivíduo adquire informação numa área de conhecimento completamente nova para ele.”.

Contudo, mapas conceituais podem ser mais facilmente entendidos a partir da sua construção, pois é neste momento que os conceitos vêm à tona e começam a fazer sentido. Em uma perspectiva mais prática pode-se definir mapas conceituais como a relação entre <CONCEITO1> <FRASE DE LIGAÇÃO> <CONCEITO2>, originando <PROPOSIÇÕES>. Um exemplo clássico é apresentado pela Figura 14.

Figura 14 - Exemplo de Mapa Conceitual



Fonte: CEMED (2010).

O objetivo de se utilizar os Mapas Conceituais nessa proposta está vinculado à necessidade de avaliar os resultados do processo, o qual ocorre, nesse caso, em dois momentos: depois da explicação expositiva dos conceitos de POO e ao final da pratica planejada por meio das unidades de estudo com o uso do Greenfoot. Assim, o ciclo “intervenção > avaliação > intervenção > avaliação” é utilizado para verificar os avanços ou problemas para alcançar os objetivos da aprendizagem de POO.

Mapas conceituais podem ser utilizados para avaliar a aprendizagem dos estudantes, pois a partir da sua construção é possível obter uma imagem da organização conceitual e das relações hierárquicas entre conceitos que o aluno

estabelece referente a um determinado conteúdo. Dessa maneira, é possível representar a aprendizagem em qualquer disciplina. Todavia, essa abordagem representa uma visão qualitativa, mas que pode ser utilizada pelo professor, para organizar a sua prática pedagógica (DA ROSA e LORETO, 2013).

Por outro lado, muitas vezes faz-se necessária a avaliação quantitativa da aprendizagem, pois o sistema educacional muitas vezes requer que a avaliação seja sintetizada por valorização numérica. Nesse sentido, Peña et al. (2005), em observância às considerações de Novak a respeito dessa temática, destacam os elementos mais significativos presentes nos mapas conceituais com possibilidade de avaliação:

> a) As *proposições*, isto é, os conceitos com as palavras de ligação apropriadas que nos indicarão as relações válidas ou equivocadas.
> b) A *hierarquização*, sempre no sentido de que os conceitos mais gerais incluam mais específicos.
> c) As *relações cruzadas*, que mostram as relações entre conceitos pertencentes a diferentes partes do mapa conceitual.
> d) Os *exemplos*, em certos casos, para assegurar que os alunos compreenderam, correspondendo à expectativa, o que é conceito e o que não é (PEÑA et al, 2005, p.132-133).

Para Novak, a pontuação quantitativa de mapas conceituais é na maioria dos casos irrelevante, pois a intencionalidade dessa técnica visa representar a estrutura cognitiva dos indivíduos, portanto devem ser analisados de forma qualitativa, todavia, o autor compreende a necessidade dos professores em quantificar o saber (NOVAK e GOWIN, 1984):

> A pontuação era, em muitos aspectos, irrelevante, uma vez que procurávamos alterações qualitativas na estrutura dos mapas conceptuais criados pelas crianças. Mas, dado que vivemos numa sociedade orientada pelos números, grande parte dos alunos e professores queriam pontuar os mapas conceptuais. Por isso, ao longo dos anos, elaborámos uma variedade de métodos de pontuação[...] (NOVAK e GOWIN, 1984, p.111).

Assim, dentro do contexto dessa pesquisa, a avaliação dos mapas conceituais produzidos pelos estudantes compõe parte do estudo, pois existe a necessidade de se quantificar os avanços da aprendizagem antes e depois das intervenções práticas. Nesse caso, foi utilizado o método proposto por Martins et al (2009), o qual pontua os elementos sugeridos por Novak, baseando-se na teoria de Ausubel. A Tabela 9 ilustra os critérios e respectivas pontuações.

Tabela 9 - Critérios avaliação dos mapas conceituais

Critérios classificatórios		**Pontuação**
Proposições (ligações entre dois conceitos) Cada ligação se for:	Válida e significativa	1
Hierarquia	Cada nível válido	5
Ligações Transversais Cada ligação se for:	Válida e significativa	10
	Somente válida	2
	Criativa ou peculiar	1
Exemplos	Cada exemplo válido	1

Fonte: Martins et al (2009) com adaptações.

O processo de avaliação dos mapas conceituais consiste na análise minuciosa, sob a ótica do pesquisador, da existência dos elementos pontuados na Tabela 9. Nesse caso, para ilustrar tal processo, foram selecionados aleatoriamente os mapas (da primeira e segunda amostra) de dois estudantes, um de cada turma participante da pesquisa, sendo que os resultados podem ser verificados nos apêndices H, I, J e K.

4.5.3 Unidades de estudo

O planejamento educacional representa um importante fundamento da prática pedagógica, pois dinâmicas de aulas são intervenções na realidade das pessoas e, portanto, precisam ser categoricamente planejadas a fim de que tenham os objetivos alcançados, com riscos e benefícios calculados. Planejar o ato educativo significa projetar o futuro, refletir previamente acerca do que se pretende realizar. Sobre essa temática, Filatro e Cairo (2015, p.225) comentam que:

> Em educação, o planejamento remete a ações previamente organizadas para responder aos desafios da aprendizagem, estabelecendo antecipadamente caminhos que norteiam a execução, o acompanhamento e a avaliação do processo educacional.

Segundo as autoras, o planejamento educacional resulta em produtos, que podem ser materializados na forma de: livros impressos e digitais, objetos de aprendizagem, *podcasts*, vídeos, infográficos, etc., sendo que cada produto deve ser elaborado com base na metodologia da gestão de projetos (FILATRO e CAIRO, 2015). Para as autoras, conteúdos e proposições de atividades devem, em essência, simular

a apresentação de conteúdos e a proposição de atividades de aprendizagem realizada do professor aos seus alunos.

Para o desenvolvimento de materiais com qualidade didática adequada, faz-se necessária a observância de fundamentos teóricos da pedagogia durante a sua elaboração. Nesse caso, Filatro e Cairo (2015) exploram dois tipos de aprendizagem apoiada por conteúdos educacionais, sendo eles: **aprendizagem que ocorre pela atenção** ou **aprendizagem que ocorre pela ação**. No primeiro caso, os conteúdos assumem natureza discursiva, pois a aprendizagem acontece em torno de ações como “assistir”, “ler” e “apreender”. Na segunda opção, os materiais utilizados exigem que os estudantes participem ativamente das atividades, pois as ações desse tipo de aprendizagem sugerem a realização de debates, investigações e experimentos práticos.

Além de discutir como o estudante aprende, é fundamental refletir como o material educacional interage com o aprendiz, pois, conforme já comentado, os conteúdos e proposições de atividades devem simular a apresentação realizada pelo professor aos seus alunos. Assim, tais materiais tendem a seguir duas abordagens distintas: **instrução direta** e **instrução indireta** (FILATRO e CAIRO, 2015).

No caso da instrução direta, o material é apresentado aos estudantes na forma de instrução, o que acontece por meio da disponibilização de materiais didáticos de diferentes naturezas (livros, tutoriais, vídeos, áudios, etc). Uma característica importante dessa abordagem é que o docente prepara os objetivos da aprendizagem e os entrega explicitamente aos estudantes. Dessa maneira, a aprendizagem ocorre em um único sentido – do professor para o aluno. Essa metodologia constantemente recebe muitas críticas por desconsiderar a existência de diversidade cognitiva verificada em um ambiente escolar, pois estudantes carregam diferentes históricos e experiências de aprendizagens o que acarreta em diferentes tempos e condições de assimilação do saber.

É importante salientar que a instrução direta não se trata apenas de uma exposição arbitrária do professor para os alunos. A essência dessa abordagem está pautada pela disponibilização sequencial de conteúdos levando-se em conta a hierarquia entre eles e, essa organização é de responsabilidade do profissional docente. Pois, cabe ao professor identificar a existência de pré-requisitos entre os conteúdos a serem apresentados de forma a facilitar a aprendizagem.

De outro lado, temos a instrução indireta onde o estudante aprende por meio de materiais que lhe proporcionem certa multiplicidade de experiências, seja pela observação, experimentação, interação com colegas ou outros atores. Nesse modelo, o estudante possui autonomia para seguir o itinerário de aprendizagem que lhe for mais adequado. Assim, são respeitadas as peculiaridades históricas e cognitivas de cada indivíduo, o que em tese, pode oferecer melhores condições de apropriação do conhecimento.

Mesmo sendo a instrução indireta um processo mais dinâmico e flexível, alguns críticos vêm deficiências na sua aplicabilidade, pois, consideram que a ausência de um mediador designado para gerenciar o processo atua como limitador da eficiência dessa abordagem.

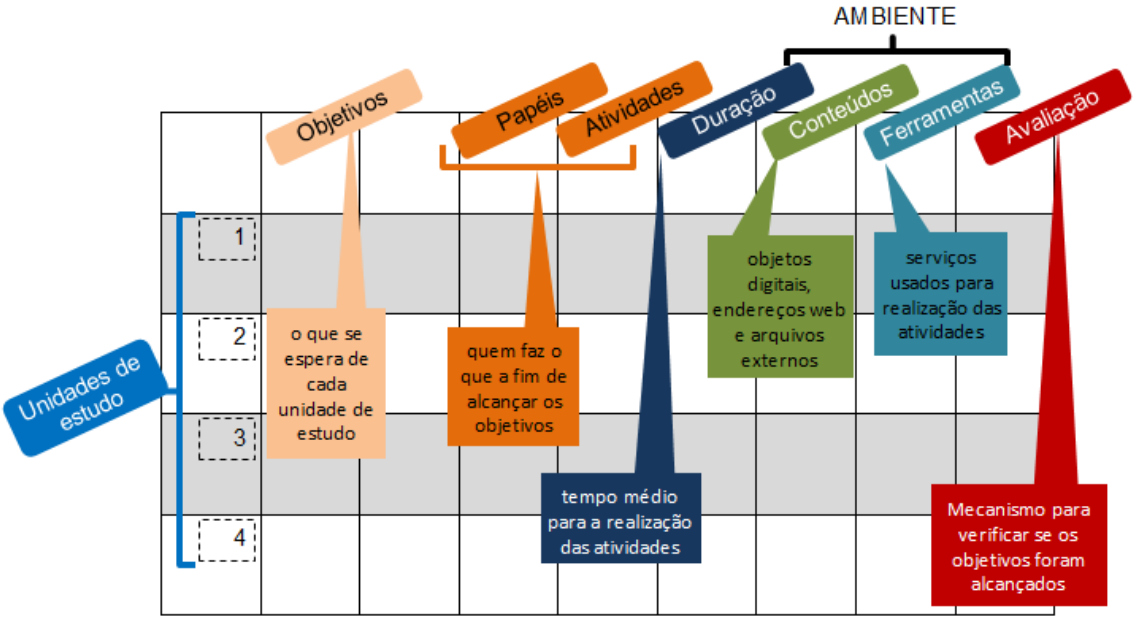
Nesse contexto, é possível verificar a existência de aspectos positivos e limites em ambas às abordagens. Dessa maneira Filatro e Cairo (2015, p. 229) apresentam a seguinte opinião:

> A nosso ver, o justo equilíbrio está na combinação de ambos os tipos de instrução, mesclando-se estrutura, diálogo e autonomia. O planejamento educacional e o design instrucional podem ser bem estruturados, particularmente no inicio da aprendizagem e/ou como fechamento de unidades de estudo, ao mesmo tempo que os próprio alunos, por meio de ferramentas cognitivas e comunicacionais, e também pela interação com outras pessoas, podem criar de forma ativa associações e prosseguir caminhos inter-relacionados, segundo suas características e seus interesses, a partir da proposição de atividades autênticas.

Assim, para a construção de um planejamento adequado e eficiente, Filatro e Cairo (2015) sugerem a utilização de um "*Metamodelo Educacional*[21]" que seja capaz de agregar princípios e teorias da pedagogia a fim de se obter melhores resultados na realização de atividades de ensino e aprendizagem. A matriz capaz de agregar uma estrutura que contemple a proposta do metamodelo educacional é ilustrada pela Figura 15.

[21] A busca por um consenso internacional, motivada também pela necessidade de representar as práticas educacionais em uma linguagem compreensível tanto por seres humanos quanto por sistemas computacionais, estimulou a criação de um **metamodelo educacional**. Esse metamodelo visa representar a variedade de teorias e aplicações na área educacional, extraindo de cada situação específica os elementos comuns de toda e qualquer ação de ensino-aprendizagem (FILATRO & CAIRO, 2015, p.230, grifo no original).

Figura 15 - Matriz de planejamento baseada em metamodelo educacional



Fonte: Filatro e Cairo (2015, p. 232, com adaptações)

A Figura 15 representa um arcabouço de aprendizagem muito semelhante ao plano de aula tradicional, porém, ao analisar atentamente verifica-se a existência de elementos especiais que proporcionam contextualização aprimorada em relação à estrutura do plano de aula convencional. É o caso das “Unidades de estudo”, que têm o papel de articular todos os elementos do metamodelo.

Unidades de estudo, segundo Filatro e Cairo (2015), podem ser aplicadas a qualquer processo de ensino e aprendizagem e seu tamanho e complexidade são definidos pelos seguintes requisitos:

a) Uma unidade não pode ser subdividida sem perder seu significado;
b) O objetivo de aprendizagem define o tamanho e a complexidade de uma unidade;
c) O tempo de execução é previamente determinado.

Os “papéis” representam outro elemento especial do metamodelo, pois, diferente do que costumamos verificar no plano de aula, existe certa dinamicidade nesse quesito, visto que os atores poderão desempenhar diferentes papéis dependendo da proposta.

> Podemos montar um grupo de alunos e propor que eles desempenhem papéis diferentes em duplas, trios, pequenos grupos, na turma, entre turmas da mesma instituição, entre instituições, no mundo... Podemos até inverter os

> papéis e colocar alunos na posição de educadores, e professores no papel de alunos (FILATRO e CAIRO, 2015, p.233).

Já o “Ambiente” é formado pelos conteúdos (ou recursos educacionais) e pelas ferramentas, organizados de maneira que componham o cenário capaz de oferecer ao estudante condições para alcançar os objetivos da proposta. Do mesmo modo, a avaliação despenha função fundamental no processo, pois resultados são constatados a partir da reflexão acerca do processo baseado na premissa “avaliar para aprender” e não “avaliar se alguém aprendeu” (FILATRO e CAIRO, 2015).

Em resumo, as unidades de estudo, para essa proposta, desempenham papel fundamental na estruturação das atividades, pois contemplam com propriedade as bases teóricas relacionadas à prática pedagógica. Nesse sentido para a realização dessa pesquisa foram construídas 6 (seis) unidades de estudo conforme descrição abaixo:

a) **Unidade de Estudo 1.1 (APÊNDICE A)**: tem como objetivos pedagógicos compreender o que são objetos computacionais, classes, atributos, métodos e instanciação, bem como entender a relação intrínseca entre classes e objetos e a importância da abstração para o desenvolvimento de software. Disponibiliza materiais anexos através de links para conteúdo que se encontra em repositório particular online (apresentação, apostila, tutorial e banco de imagens), bem como um pequeno texto envolvendo os principais conceitos abordados na unidade. Nessa unidade de estudo, os estudantes devem observar a imagem ilustrativa e um vídeo do protótipo do simulador de um veículo e perceber quais são os objetos existentes no contexto, suas características e ações. Na sequência, devem descrever as classes de tais objetos em diagramas UML com apoio da ferramenta Draw.io e, com o apoio do Greenfoot, construir cenário semelhante ao visualizado, mas sem a necessidade de fazê-lo funcionar. Com essas atividades busca-se desenvolver nos estudantes a percepção de que o mundo é formado por objetos e que estes são descritos nas classes, ainda, que a construção de um software passa, primeiramente, por um momento de planejamento – abstração – materializado pela construção de diagramas UML;
b) **Unidade de Estudo 1.2 (APÊNDICE B)**: os objetivos dessa unidade de estudos são de compreender os conceitos de método, atributo,

instanciação, mensagem e construtores. É uma continuidade da unidade de estudo 1.1, embora seja possível desenvolver as atividades nela descritas de maneira independente. Consiste na construção parcial do cenário de um simulador de carro que, nesta unidade, necessita apenas fazer a pista se movimentar – o movimento da pista depende da instanciação de dois objetos do tipo “Pista” que se comunicam entre si, consultando suas posições no plano cartesiano, a fim de exemplificar como ocorre o envio e o recebimento de mensagens entre objetos e a importância dos construtores no processo. Atributos são declarados, métodos são programados e utilizados possibilitando que o estudante reflita sobre o funcionamento destes elementos na execução de um software;

c) **Unidade de Estudo 1.3 (APÊNDICE C)**: a unidade de estudo 1.3 tem a finalidade principal de abordar o conceito de encapsulamento. Tem também a intencionalidade de reforçar os conceitos abordados nas unidades 1.1 e 1.2 pelo desenvolvimento integral do projeto “simulador” que desenvolve o prospecto de um simulador de carro utilizando o Greenfoot. A intenção é construir o cenário completo e funcional a partir da modelagem disponibilizada e descrições literais do funcionamento de cada classe e método. Nesse caso, muitos dos conceitos são retomados, tais como: classe, atributo, método, construtor, mensagem, instanciação, objeto, etc. Para além destes, os atributos são encapsulados e, nesse caso, é possível trabalhar questões relacionadas a padrões de acesso e métodos assessores. Todos esses conceitos são abordados sob uma perspectiva prática, em que os estudantes têm os elementos descritos e necessitam implementá-los na ferramenta. Ainda, para criar um estímulo a mais, os estudantes são instigados a pensar funcionalidades extras por meio do “desafio” da criação de uma classe não descrita, chamada “Obstáculo”. Nesse caso, abre-se espaço para que usem a sua criatividade e realizem pesquisas autônomas para alcançar o objetivo;

As três unidades descritas acima, conforme já mencionado, foram desenvolvidas para a intervenção piloto, realizada com a primeira turma em 2017. Já as três unidades descritas a seguir foram desenvolvidas com base na avaliação da

primeira intervenção sendo que, conforme já informado anteriormente, foram executadas as unidades 2.1 e 2.3 com a segunda turma, em 2018. São elas:

d) **Unidade de Estudo 2.1 (APÊNDICE D)**: tem como objetivo compreender os conceitos elementares da POO: objetos, classes, atributos, métodos e instanciação. Além disso, busca oferecer espaço para a compreensão da relação entre classe e objeto, a finalidade dos construtores e da comunicação entre objetos. São disponibilizados materiais armazenados em repositórios online para servir de material de apoio. A unidade de estudo apresenta, inicialmente, um breve texto contextualizando os principais conceitos abordados em seu escopo e na sequência propõe a realização de uma atividade envolvendo a construção de um protótipo de jogo com o uso do Greenfoot. Trata-se de um jogo muito básico, onde que o ator principal (*man*) necessita atravessar uma movimentada avenida e chegar até o mercado. Nesse cenário, os carros surgem aleatoriamente na margem esquerda da tela e trafegam para a direita em velocidades diferentes uns dos outros. O ator principal é comandado pelas setas direcionais e precisa desviar dos veículos, pois, caso colida com algum carro, ele perde (Game Over) – caso consiga chegar ao “mercado”, o jogador vence.
e) **Unidade de Estudo 2.2 (APÊNDICE E)**: a unidade de estudo 2.2 tem como finalidade ratificar os conceitos básicos de POO (classe, método, atributo, instanciação, construtor, envio de mensagem e objeto) e, dedicar atenção especial no que se refere ao conceito de encapsulamento e conteúdos relacionados (métodos acessores e padrões de acesso). A unidade de estudos inicia sua apresentação disponibilizando materiais auxiliares (apostila, apresentação, tutorial e banco de imagens) – hospedados em links online com acesso para leitura. Na sequência, um breve texto abordando os conceitos envolvidos é disponibilizado para dar suporte ao desenvolvimento das atividades propostas. Em seguida, os estudantes são desafiados em produzir, com o auxílio do Greenfoot, um simulador de um veículo, em que as setas direcionais exercem as funções básicas desse veículo (acelerar, frear, virar à esquerda ou à direita). O protótipo conta com um velocímetro que marca a velocidade atual do carro e na lateral esquerda, há um contador de tempo e de distância percorrida. O carro precisa, para

vencer o jogo, percorrer 5000(cinco mil) metros sem colidir com os obstáculos que vão surgindo aleatoriamente sobre a pista, pois, caso colida com algum desses obstáculos, o jogador perde.

f) **Unidade de Estudo 2.3 (APÊNDICE F)**: tem como finalidade proporcionar ao estudante a percepção de como os conceitos de POO se relacionam a fim de possibilitar a construção eficiente de um produto de software. Assim, esta unidade de estudo busca proporcionar o entendimento dos conceitos elementares desse paradigma de desenvolvimento e, dar suporte à compreensão do conceito de herança e sua relação com o polimorfismo. Para que estes objetivos sejam alcançados, essa unidade de estudo apresenta inicialmente 8 (oito) premissas, devidamente fundamentadas, acerca dos principais conceitos de POO. Como atividade, é proposta a construção de um protótipo de jogo com apoio do Greenfoot. Tal protótipo é chamado de “Colheita das Maçãs” e é organizado em um plano 2D onde o ator principal (herói), representado pela figura de uma ‘joaninha’, precisa colher maçãs espalhadas aleatoriamente no cenário. Cada maçã colhida lhe rende um ponto e, para vencer, precisa alcançar 10 (dez) pontos. O ‘herói’ tem o desafio, para além de colher as maçãs, desviar-se de asteroides que caem aleatoriamente, pois, caso colida com algum deles, o jogador perde. Para que os conceitos de herança e polimorfismo fiquem evidentes, há no jogo a classe *SuperMaca.class,* que é uma especialização da classe *Maca.class*. Tal classe (SuperMacas) é representada no jogo por maçãs verdes, maiores que as demais, e que cruzam aleatoriamente o cenário, sendo que, caso o ‘herói’ consiga colher uma delas, é promovido ao status de ‘SuperHeroi’ com imagem diferente e métodos reimplementados – comportamento diferente na colheita das maçãs, porém, ainda preservando todos as demais características da classe ancestral.

4.6 EMBASAMENTO PEDAGÓGICO: CICLO DE APRENDIZAGEM DE KOLB

O embasamento pedagógico da prática é norteado pela proposta do *Ciclo de Aprendizagem de Kolb*, o qual foi desenvolvido na década de 1980 por David Kolb. O

referido ciclo é parte integrante da sua teoria da aprendizagem experiencial[22] (KOLB, 1984), a qual busca elencar as diferentes formas de aprender dos indivíduos. Nesse sentido, o autor considera que a aprendizagem ocorre por meio de um ciclo contínuo composto de quatro estágios:

a) Experiência Concreta (agir);
b) Observação Reflexiva (refletir);
c) Conceitualização Abstrata (conceitualizar);
d) Experimentação Ativa (aplicar).

A Tabela 10 descreve cada um dos estágios desse círculo, elencando as principais ações características de cada um.

Tabela 10 - Estágios do processo de aprendizagem segundo David Kolb

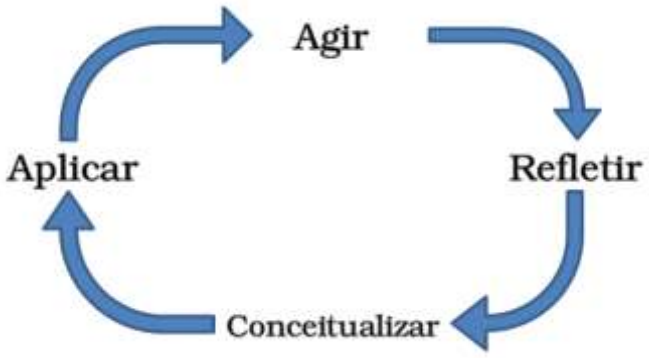
ESTÁGIO	DESCRIÇÃO
Experiência Concreta	Os estudantes têm experiências relacionadas a fazer uma tarefa. Eles trabalham com uma nova experiência concreta, tendendo a tratar as situações mais em termos de observações e sentimentos do que com uma abordagem teórica e sistemática.
Observação Reflexiva	Os estudantes estão envolvidos em observar, revendo e refletindo sobre a experiência concreta do estágio anterior. As reflexões e observações neste estágio não incluem necessariamente realizar alguma ação.
Conceitualização Abstrata	Neste estágio os estudantes se desenvolvem e agem no domínio cognitivo da situação usando teorias, hipóteses e raciocínio lógico para modelar e explicar os eventos.
Experimentação Ativa	Os estudantes estão envolvidos em atividades de planejamento, experimentando experiências que envolvem mudança de situações. Os estudantes usam as teorias para tomar decisões e resolver problemas.

Fonte: Marietto et al. (2014, p.528).

Dessa maneira, a teoria experiencial proposta por Kolb indica, a exemplo de outros teóricos da pedagogia, que a aprendizagem se dá por meio de um processo contínuo e evolutivo, o qual possibilita ao indivíduo ir do *não saber* até o *saber*. Para Kolb tal processo consiste na progressão do estudante no círculo composto pelos quatros estágios: agir, refletir, conceitualizar e aplicar.

[22] A teoria da Aprendizagem Experiencial proposta por David Kolb em 1984 defende que o processo pelo qual o conhecimento é criado se dá através da transformação da experiência. Embora essa teoria tenha sido, em primeira análise, criada para a qualificação profissional, pode ser aplicada em outras situações que envolvam processos de ensinar e aprender.

Figura 16 - Ciclo de aprendizagem de Kolb



Fonte: do autor.

Esse “cíclo de aprendizagem” é utilizado por Kolb(1984) como princípio central de sua teoria, onde ‘experiências imediatas ou concretas’ fornecem elementos necessários para “observações e reflexões”. Dessa maneira, as referidas “observações e reflexões” são transformadas em “conceitos abstratos”. Tais conceitos possibilitam que o indivíduo realize testes acerca das suas constatações, criando possibilidades de novas experiências ativas (PIMENTEL, 2007).

Nesse sentido, este trabalho apropriou-se do embasamento teórico proposto por Kolb, na utilização do desenvolvimento de jogos para facilitar a aprendizagem dos conceitos de POO, pois, ao desenvolver jogos, todas as fases do “ciclo de aprendizagem de Kolb” podem ser contempladas. Isso é possível em função da forma com a qual a proposta das atividades é apresentada aos estudantes, permitindo que eles tenham um contato inicial com a “experiência concreta” para que possam “observar e refletir” acerca dos conceitos ali implícitos. De outro lado, esse movimento permite que abstraiam suas conceitualizações e possam testá-las, o que lhes oferece experimentações e nova experiência concreta.

Esse capítulo apresentou os materiais e métodos utilizados na pesquisa, de forma a proporcionar o entendimento dos procedimentos e recursos norteadores do processo. O próximo capítulo tem a finalidade de descrever os resultados obtidos a partir das intervenções.

5 POSSIBILIDADES E LIMITES DA UTILIZAÇÃO DO DESENVOLVIMENTO DE JOGOS NO PROCESSO DE ENSINO E APRENDIZAGEM DE POO

O desenvolvimento da proposta teve início, conforme planejado, no início do mês de novembro de 2017 com a execução da “Intervenção Piloto” junto à turma de concluintes do curso Técnico em informática Integrado ao Ensino Médio de 2017. A turma era composta de 26 (vinte e seis) estudantes, no entanto, apenas 18(dezoito) participaram ativamente de todas as etapas propostas na pesquisa, dessa maneira, no que se refere à tabulação e análise dos mapas conceituais (item ‘f’), apenas os dados dos referidos 18 estudantes foram analisados. A segunda intervenção aconteceu nos meses de março a junho de 2018, com os estudantes do terceiro ano do mesmo curso, composta de 20 (vinte) estudantes.

Para preservar o anonimato dos estudantes participantes da pesquisa, seus nomes foram ocultados e substituídos pela sequência, no caso da intervenção piloto, de A1 a A26 e, no caso da segunda intervenção, os nomes foram substituídos pela sequência E1 a E20. Nas próximas seções os dados obtidos a partir do desenvolvimento da proposta serão detalhados.

5.1 INTERVENÇAO PILOTO (2017)

De acordo com as etapas da pesquisa, elencadas na Seção 4.4.1, a “intervenção piloto” foi desenvolvida entre os meses de novembro a dezembro de 2017 e, conforme planejamento, a primeira atividade consistiu na apresentação/retomada dos conceitos de POO aos estudantes envolvidos.

Concluída essa primeira atividade, a sequência da intervenção determina a construção individual de mapas conceituais sobre a temática “Programação Orientada a Objetos”. Nesse caso, inicialmente foi necessário explicar aos estudantes a dinâmica utilizada na construção de tais mapas, para que pudessem compreender como são construídos a partir da relação entre <*conceito1*> <*frase de ligação*> <*conceito2*>, originando <*proposições*>. A dinâmica foi um tanto facilitada pela utilização da ferramenta Cmap Tools[23], pois foi especialmente desenvolvida para essa

[23] Ferramenta Cmap Tools pode ser obtida em http://cmap.ihmc.us.

finalidade. Foi necessário, no entanto, disponibilizar no enunciado da atividade, de maneira desvinculada dos principais conceitos, para que a partir disso os estudantes conseguissem se concentrar nas relações existentes entre eles. De modo geral, a atividade de construção dos mapas conceituais transcorreu bem, pois mediante pequenos auxílios todos conseguiram concluir essa atividade.

Na sequência, seguindo os procedimentos da intervenção piloto, em 13 (treze) horas/aula foi proposto aos estudantes do desenvolvimento das unidades de estudo 1.1, 1.2 e 1.3, detalhadas nos Apêndices A, B e C. As considerações acerca do andamento desta atividade estão descritas na seção 5.4.

Conforme planejamento, no passo seguinte da proposta, os estudantes refizeram os mapas conceituais sob a temática “Programação Orientada a Objetos” e responderam ao questionário de opinião (APÊNDICE G), composto de quatro perguntas fechadas e uma para livre manifestação acerca das atividades desenvolvidas. Os dados obtidos nestas atividades estão tabulados nas seções 5.1.1 e 5.1.2.

5.1.1 Produção de Mapas Conceituais – Turma 2017 (Piloto)

Essa seção tem o objetivo de apresentar a tabulação e análise dos dados obtidos a partir da construção dos mapas conceituais previstos nos itens ‘b’ e ‘d’ descritos na seção 4.4.1. Nesse caso, é importante salientar que essa atividade tem por objetivo apresentar uma análise comparativa entre a primeira e a segunda coleta a fim de verificar possíveis avanços na estrutura cognitiva dos estudantes a partir da realização das atividades de desenvolvimento de jogos para aprendizagem de POO.

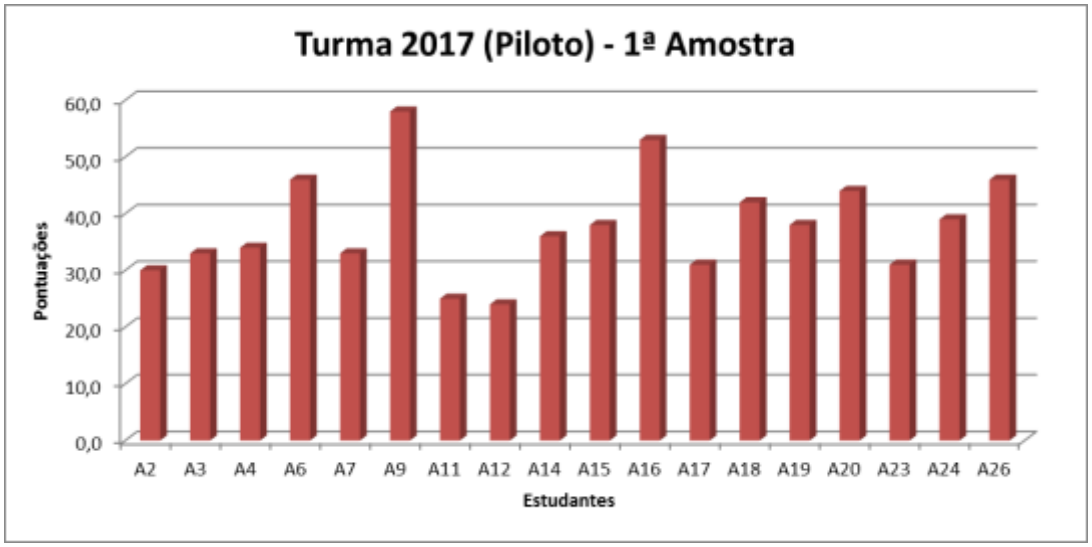
Conforme descrito na seção 4.5.2, os mapas foram avaliados conforme critérios definidos por Martins et al (2009), observando os critérios:

a) Proposições válidas;
b) Níveis hierárquicos;
c) Ligações transversais;
d) Exemplos[24].

[24] No caso dos exemplos, segundo Novak e Gowin (1984), fazem referência à apresentação de exemplos concretos acerca de determinado conceito ou grupo de conceitos, nesse caso, a sua elaboração foi opcional na atividade de construir o seu mapa conceitual.

As pontuações para cada um dos critérios estão definidas na Tabela 9. O Gráfico 4 ilustra a pontuação obtida pelos estudantes da turma na primeira amostra.

Gráfico 4 – Avaliações dos Mapas Conceituais- Turma Piloto (1ª amostra)



Fonte: do autor.

Em análise às pontuações obtidas pelos estudantes nessa primeira amostra, constataram-se os valores descritos na Tabela 11.

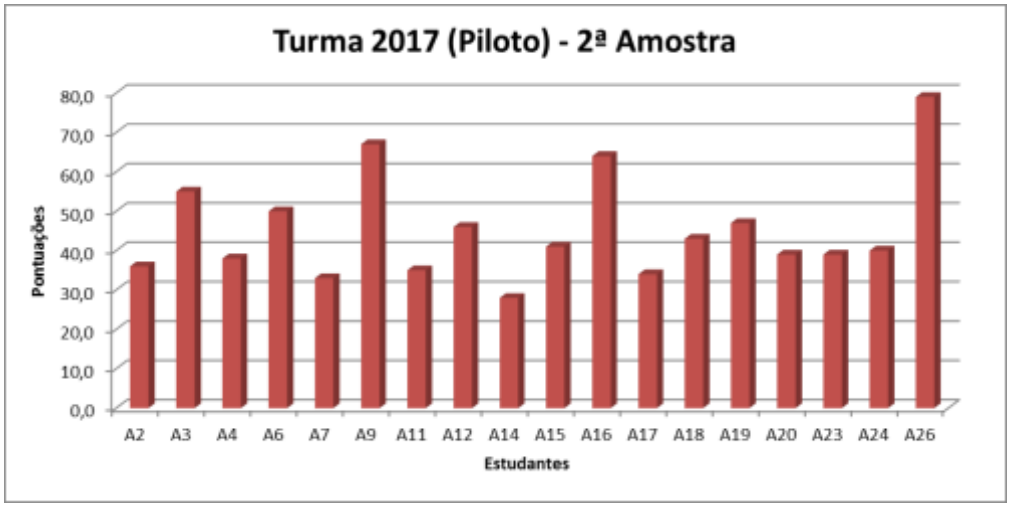
Tabela 11 - Análise das avaliações dos Mapas Conceituais- Turma Piloto (1ª amostra)

Turma 2017 (Piloto) - 1ª Amostra		
Média pontuação	37,83	
Maior nota	58,00	
Menor nota	24,00	
Desvio padrão	8,83	
Acima da média	9	50%
Abaixo da média	9	50%
Acima da média e do desvio padrão	2	11%
Abaixo da média e do desvio padrão	2	11%

Fonte: do autor.

Depois de realizadas as atividades descritas nos Apêndices A, B e C os estudantes reconstruíram os mapas conceituais a fim de que pudéssemos analisar possíveis avanços na compreensão dos conceitos de POO. Assim, os dados obtidos a partir dessa segunda amostra são explícitos no Gráfico 5.

Gráfico 5 - Avaliações dos Mapas Conceituais- Turma Piloto (2ª amostra)



Fonte: do autor.

Do mesmo modo que da primeira amostra, os dados foram tabulados e apresentados na Tabela 12.

Tabela 12 - Análise das avaliações dos Mapas Conceituais- Turma Piloto (2ª amostra)

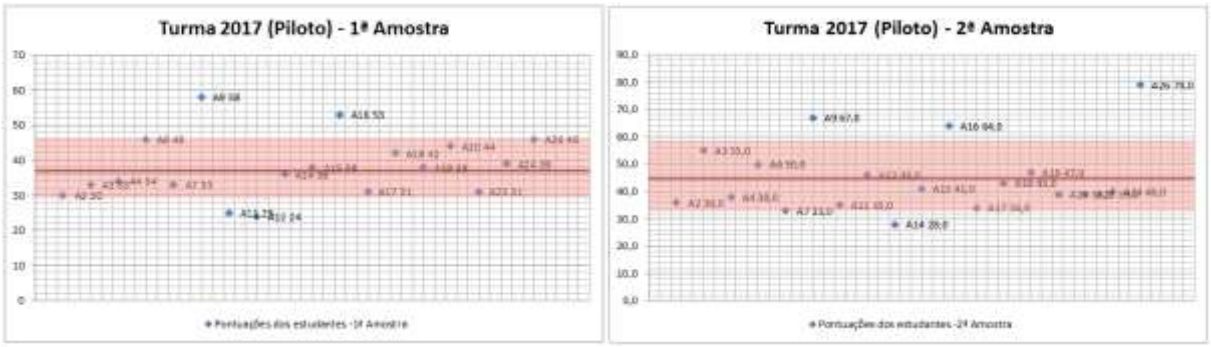
Turma 2017 (Piloto) – 2ª Amostra		
Média pontuação	45,22	
Maior nota	79,00	
Menor nota	28,00	
Desvio padrão	12,98	
Acima da média	7	39%
Abaixo da média	11	61%
Acima da média e do desvio padrão	3	17%
Abaixo da média e do desvio padrão	1	6%

Fonte: do autor.

Em um comparativo entre a primeira e segunda coleta na turma piloto (Tabela 13), verificaram-se pontos positivos e negativos, pois, constata-se que a média geral da turma subiu (em pontos) de 37,82 para 45,22. A maior nota vai de 58,00 pontos na primeira amostra para 79,00 na segunda coleta. Verificou-se também que a menor nota subiu de 24,00 para 28,00. De outro lado, o desvio padrão[25] da turma subiu de 8,83 para 12,98 o que demonstra que na segunda coleta as pontuações ficaram mais dispersas em relação à média geral (Figura 17).

[25] O desvio padrão mede a dispersão, considerando o quanto as observações se afastam de sua média (MOORE, 2005, p.37).

Figura 17 - Comparativo entre Gráficos de Dispersão entre a 1ª e 2ª Amostras – Turma 2017



Fonte: do autor.

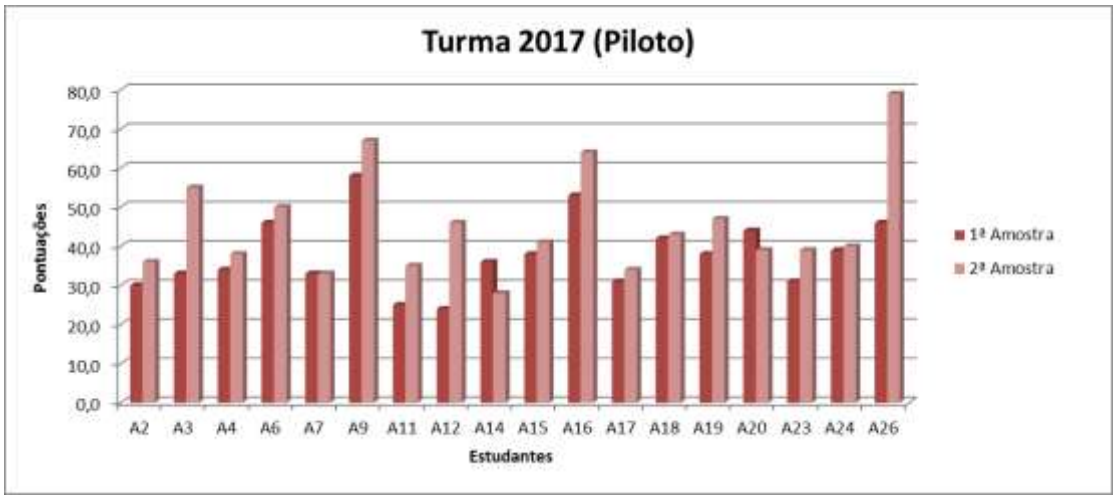
Outro ponto negativo foi que o número de estudantes acima da média caiu de 50% na primeira amostra para 39% na segunda.

Tabela 13 - Comparativo entre primeira e segunda amostra (Turma Piloto)

Comparativo primeira X segunda amostra (Turma Piloto)		
Item Observado	**Primeira amostra**	**Segunda amostra**
Média	37,83	45,22
Maior nota	58,00	79,00
Menor nota	24,00	28,00
Desvio padrão	8,83	12,98
Acima da média	9 (50%)	7 (39%)
Abaixo da média	9 (50%)	11 (61%)
Acima da média e do desvio padrão	2 (11%)	3 (17%)
Abaixo da média e do desvio padrão	2 (11%)	1 (6%)

Todavia, talvez seja mais importante nesse contexto, a constatação dos avanços individuais dos estudantes, pois apenas dois estudantes (11%) apresentaram piora na pontuação na segunda coleta – caso dos estudantes A14 e A20, sendo que dos demais, três (16%) mantiveram suas pontuações muito próximas à primeira amostra (A7, A18, A24) e os demais (72%) aumentaram suas pontuações, conforme se pode visualizar no Gráfico 6 e Tabela 14.

Gráfico 6 - Avaliações dos Mapas Conceituais- Turma Piloto (comparativo entre 1ª e 2ª amostra)



Fonte: do autor.

A Tabela 14 ilustra os avanços individuais de cada estudante, medido em percentuais de aumento ou retrocesso da primeira para a segunda amostra.

Tabela 14 - Comparativo individual entre 1ª e 2ª amostra (Turma 2017)

Turma 2017(Piloto)			
Aluno	**1ª Amostra**	**2ª Amostra**	**Diferença entre 1ª e 2ª Amostra (%)**
A2	30,0	36,0	20%
A3	33,0	55,0	67%
A4	34,0	38,0	12%
A6	46,0	50,0	9%
A7	33,0	33,0	0%
A9	58,0	67,0	16%
A11	25,0	35,0	40%
A12	24,0	46,0	92%
A14	36,0	28,0	-22%
A15	38,0	41,0	8%
A16	53,0	64,0	21%
A17	31,0	34,0	10%
A18	42,0	43,0	2%
A19	38,0	47,0	24%
A20	44,0	39,0	-11%
A23	31,0	39,0	26%
A24	39,0	40,0	3%
A26	46,0	79,0	72%

Fonte: do autor.

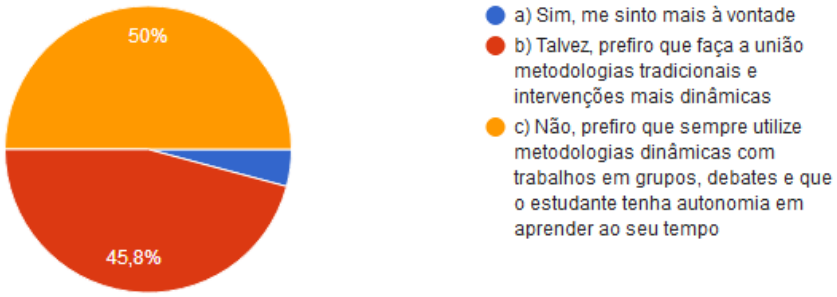
5.1.2 Questionário de opinião – Turma 2017 (Piloto)

Essa seção destina-se à tabulação e análise dos dados obtidos no questionário de opinião (APÊNDICE G) respondido pelos estudantes depois de realizar as atividades previstas na seção 4.4.1.

A primeira questão teve por objetivo mapear as preferências de dinâmicas apresentadas nas aulas, a fim de compreender o perfil da turma quanto ao perfil de aprendizagem. O Gráfico 7 ilustra os resultados obtidos a partir do questionamento: *Ao estudar conceitos de POO, prefere que o professor utilize uma metodologia tradicional, com o uso de apostila, slides e exercícios?*

Pode-se perceber que a grande maioria (95,8%) prefere atividades dinâmicas ou a união destas com aulas expositivas. Apenas um estudante (4,2%) prefere aulas puramente expositivas.

Gráfico 7 - Preferência dos estudantes quanto à metodologias empregadas pelos docentes (Turma Piloto)

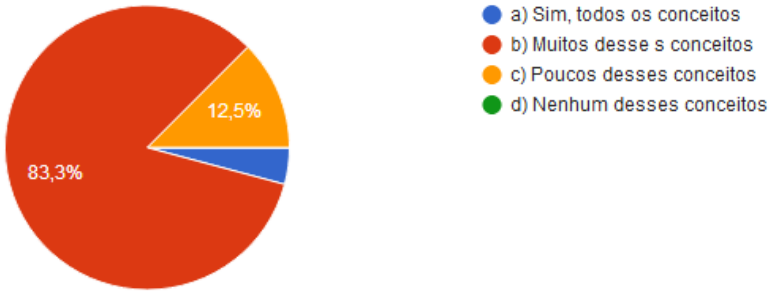


Fonte: do autor.

A segunda questão é relacionada às sensações da turma quanto ao ambiente Greenfoot para aprendizagem de POO. Dessa maneira, foram questionados: *Sobre a ferramenta Greenfoot, você conseguiu perceber a utilização dos principais conceitos de orientação à objetos (classe, objeto, instanciação, construtor, encapsulamento, herança, mensagem e polimorfismo)?* A partir das opiniões dos estudantes, pode-se constatar que a grande maioria deles considerou a ferramenta positiva para a aprendizagem de POO (Gráfico 8), sendo que 87,5% dos estudantes consideram que ‘muitos’ ou ‘todos’ os conceitos de POO podem ser visualizados a partir da utilização

da ferramenta e três estudantes (12,5%) conseguiram visualizar ‘poucos dos conceitos’ de POO.

Gráfico 8 – Percepção dos estudantes da utilização dos conceitos de POO ao utilizar Greenfoot no desenvolvimento das atividades (Turma Piloto).

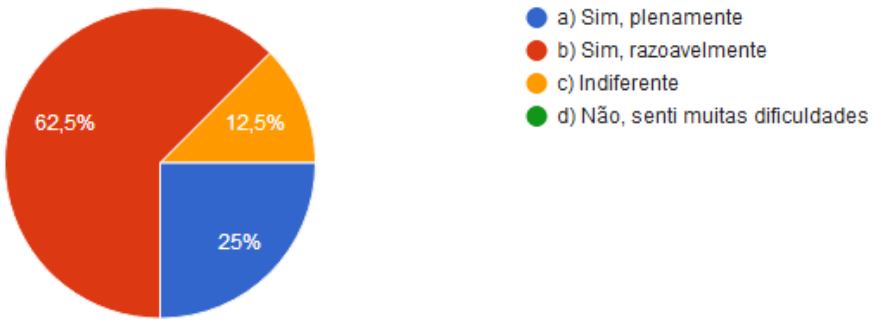


Fonte: do autor.

A terceira pergunta teve como objetivo verificar se, do ponto de vista dos estudantes, a aprendizagem dos conceitos de POO foi facilitada pelo uso do Greenfoot. Nesse caso, foram questionados: *Na sua opinião, a utilização da ferramenta Greenfoot facilitou o entendimento dos conceitos de POO?*

O Gráfico 9 demonstra que a grande maioria dos estudantes (87,5%) considerou que a partir da utilização da referida ferramenta, a aprendizagem de POO foi “razoavelmente” ou “plenamente” facilitada – 12,5% dos estudantes consideram que o seu uso é “indiferente” para facilitar a aprendizagem de POO.

Gráfico 9 – Opinião dos estudantes sobre utilizar o Greenfoot para facilitar a aprendizagem de POO (Turma Piloto)



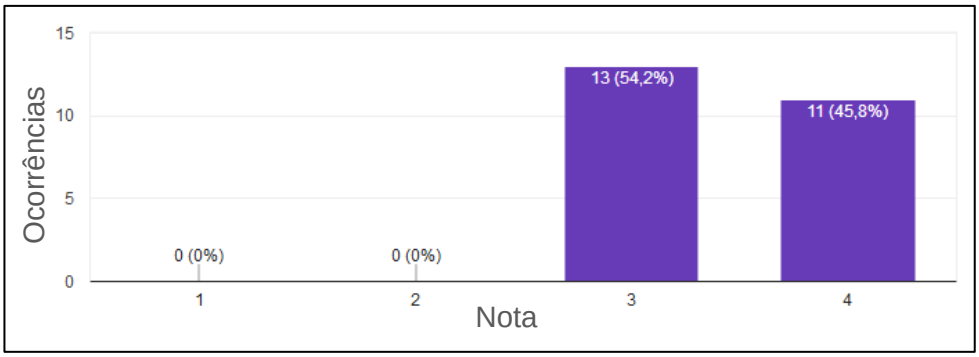
Fonte: do autor.

Três estudantes (A9, A21 e A22) assinalaram a opção “indiferente” para a terceira questão e, ao analisar as demais respostas, constata-se que A9 considera ter verificado “muitos dos conceitos de POO” no uso do Greenfoot e os demais (A21 e

A22) afirmaram ter visualizado “poucos dos conceitos de POO” na ferramenta. Já em relação às notas dos mapas conceituais, apenas o estudante A9 realizou todas as atividades obtendo nota acima da média em ambas as amostras.

A quarta questão mede, em uma escala de 1 a 4, uma nota referenciada por cada estudante sobre as dinâmicas realizadas com o apoio das Unidades de estudo e Greenfoot. Nesse sentido os estudantes foram questionados: *Se você fosse atribuir uma nota para as atividades que envolveram o ensino de POO com o uso do Greenfoot, que nota daria?*

Gráfico 10 - Notas atribuídas pelos estudantes para as atividades realizadas (Turma Piloto)

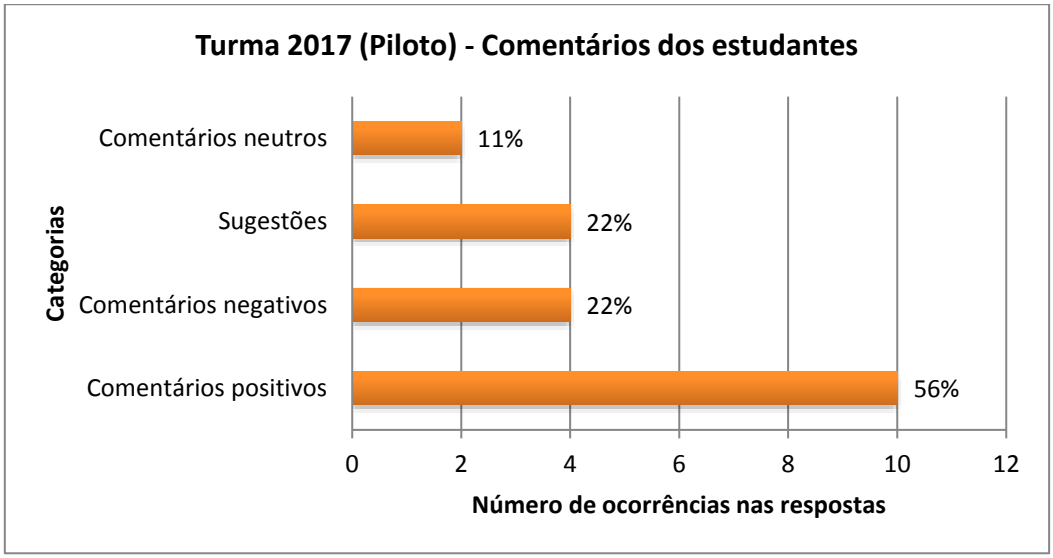


Fonte: do autor.

Pode-se perceber pelo Gráfico 10, que 45,8% dos estudantes deram a nota máxima (4) para as atividades e 54,2% avaliaram com a segunda nota mais alta (3), tais resultados podem inferir que as atividades foram positivas, pois nenhum dos estudantes apontou as notas mais baixas (1 e 2).

A última questão respondida pelos estudantes após a realização das atividades envolvendo Unidades de estudo e Greenfoot foi de livre manifestação, para que pudessem opinar de maneira aberta sobre as suas sensações quanto às atividades desenvolvidas. Nesse caso, as respostas foram categorizadas de acordo com seu conteúdo, classificado em “Comentários positivos”, “Comentários negativos”, “Sugestões” ou “Comentários neutros”. O Gráfico 11 ilustra a distribuição das respostas por categorias - sendo que em uma resposta pode haver mais de uma categoria.

Gráfico 11 - Comentários dos Estudantes (Turma Piloto)



Fonte: do autor.

Ao analisar os comentários dos estudantes, no que se refere às “Sugestões”, pode-se levantar os mais relevantes. Nesse sentido, as contribuições dos estudantes A7 e A23 sugerem que se atribua maior carga horária para o desenvolvimento das atividades.

> [...] talvez pudesse ser começado um pouco antes, por que muitas vezes não dava tempo pra fazer todas as atividades, por motivos que ainda é final do ano e tem muita prova, trabalhos e exames (A7).
>
> [...] achei um pouco cansativo o número de exercícios, mas deu para aprender muita coisa ao longo do ano (A23).

De outro lado, o estudante A18 sugere “desafiar a criação de jogos sozinho com o uso do Greenfoot”. Assim, a proposta de criação dos jogos deveria ser, segundo o estudante A18, de iniciativa de cada aluno. Nesse caso, é importante ressaltar que, dessa forma, não há garantia de que os objetivos pedagógicos sejam atendidos nas propostas dos estudantes, visto que não lhes compete tal responsabilidade e habilidade. De todo modo, a Unidade de Estudos descrita no APÊNDICE C propõe um desafio em que os estudantes podem programar livremente obstáculos para o ator principal do jogo.

Os comentários negativos foram quatro no total, seguiram três linhas diferentes. Um deles esteve relacionado ao posicionamento do docente durante as atividades na distribuição de auxílios aos estudantes.

> Acredito que tem que dar mais a atenção como um todo a turma, não apenas para alguns da turma (A8).

Outros dois comentários foram relacionados às dificuldades encontradas na utilização da ferramenta Greenfoot:

> O Greenfoot é bem complicado de entender. Porém consegui aprender um pouco, mas no Netbeans aprendi melhor (A12).
>
> Aprendi mais com o Netbeans e com o software do estágio, tive certas dificuldades na plataforma (A22).

Por fim, o comentário do estudante A25 faz uma crítica construtiva muito pertinente à proposta apresentada aos estudantes, pois sugere que o desenvolvimento do Projeto Simulador (Apêndices A, B e C) não abrigam um objetivo para o jogo desenvolvido – Nesse caso, é pertinente considerar a constatação do estudante, tendo em vista que na proposta original não existe uma "pontuação" que denote algum objetivo a ser alcançado ou um desafio a ser vencido. Tal situação foi contornada para a segunda intervenção.

Em análise às demais respostas dos estudantes que realizaram "comentários negativos", verifica-se que o estudante A8 afirma se sentir "mais à vontade" com dinâmicas expositivas, conseguiu visualizar "muitos dos conceitos de POO" nas atividades com Greenfoot, assinalou que a ferramenta facilitou "razoavelmente" o aprendizado e atribuiu nota '3' para o processo global. Os estudantes que indicaram ter tido dificuldades com o uso do Greenfoot responderam conforme Tabela 15.

Tabela 15 - Respostas dos estudantes que tiveram dificuldades com o Greenfoot (Turma Piloto)

Estudante	Q1	Q2	Q3	Q4
A12	Prefere sempre metodologias dinâmicas	Visualizou muitos conceitos de POO	A ferramenta **facilitou razoavelmente** o aprendizado	Nota 3
A22	Prefere união de metodologias dinâmicas e expositivas	Visualizou poucos conceitos de POO	O uso do Greenfoot foi **indiferente** para facilitar o aprendizado	Nota 3

Fonte: do autor.

O estudante A25 que sentiu "falta de objetivo no jogo" afirmou que prefere "sempre a utilização de metodologias dinâmicas", conseguiu visualizar "todos os conceitos de POO" nas atividades, considerou que o Greenfoot facilitou "plenamente" o aprendizado e atribuiu nota "4" para o processo global.

Já os comentários positivos estiveram em 56% das opiniões, indicando que a maioria dos estudantes gostou da proposta. Dentre os comentários positivos, podem ser destacados:

> Foram interessantes, pois foi possível ver na prática alguns conceitos de POO (A6).

> Foram boas, pois usaram uma metodologia diferente da que estava sendo utilizada (A13).
>
> Foi interessante pois com o auxílio do Greenfoot foi possível ver na prática os conceitos aprendidos, tornando assim o entendimento mais facilitado (A15).
>
> O greenfoot nos faz ver na prática os conceitos da POO (A24).

Frente a esses comentários, pode-se verificar o potencial do ambiente de programação Greenfoot para o ensino e aprendizagem de POO, pois tem a capacidade de demonstrar na prática os conceitos de POO e, ainda, agrega a dinâmica de desenvolvimento de jogos no processo.

5.2 SEGUNDA INTERVENÇÃO TURMA (2018)

As intervenções com a segunda turma iniciaram-se, conforme cronograma, no mês de março de 2018. A turma do terceiro ano do Curso Técnico em Informática Integrado ao Ensino Médio do IFFAR *Campus* Santo Augusto é formada de vinte estudantes, sendo que todos participaram ativamente dos passos previstos na seção 4.4.2. Assim, as primeiras atividades de 2018 foram relacionadas à apresentação dos conceitos de POO para que os estudantes tivessem um primeiro contato com o conteúdo e a partir disso tivessem possibilidades de realizar as demais atividades previstas na intervenção.

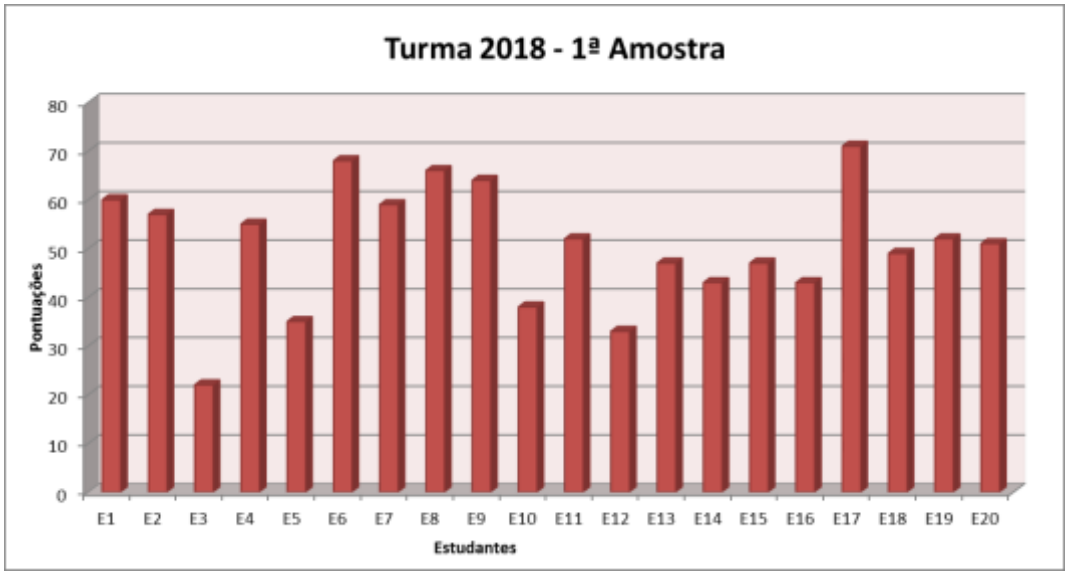
Em continuidade à proposta, os estudantes construíram os Mapas Conceituais, realizaram as atividades propostas na seção 4.4.2, refizeram o mapa conceitual com a temática POO e responderam ao questionário de opinião (APÊNDICE G) cujos dados obtidos são descritos na próxima seção.

5.2.1 Produção de Mapas Conceituais – Turma 2018

A produção dos Mapas Conceituais com a turma de 2018 aconteceu nos meses de maio e junho. Utilizou-se a mesma dinâmica da intervenção piloto, no que se refere à disponibilização dos conceitos chave de POO de maneira desvinculada, para que, a partir deles, os estudantes pudessem se concentrar nas suas relações e formar proposições. Utilizou-se também a ferramenta CmapTools para a auxiliar na tarefa de criação dos referidos mapas.

A partir da primeira amostra com a turma 2018, obtiveram-se os resultados ilustrados pelo Gráfico 12:

Gráfico 12 - Avaliações dos Mapas Conceituais- Turma 2018 (1ª amostra)



Fonte: do autor.

A análise dessa amostra está descrita na Tabela 16.

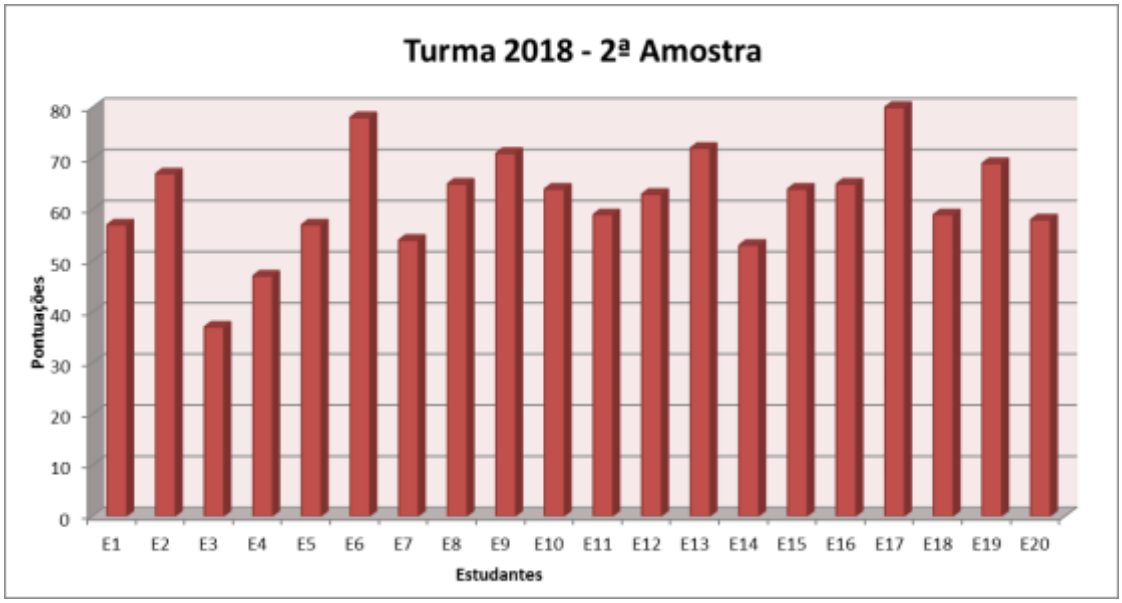
Tabela 16 - Análise das avaliações dos Mapas Conceituais- Turma 2018 (1ª amostra)

Turma 2018 (1ª amostra)		
Média da turma	50,60	
Maior nota	71,0	
Menor nota	22,0	
Desvio padrão	12,31	
Acima da média	11	55%
Abaixo da Média	9	45%
Acima da média e do desvio padrão	4	20%
Abaixo da média e do desvio padrão	3	15%

Fonte: do autor.

Depois de realizadas as atividades descritas nos Apêndices A, B e C os estudantes reconstruíram os mapas conceituais a fim de que pudéssemos analisar possíveis avanços na compreensão dos conceitos de POO. Assim, os dados obtidos a partir dessa segunda amostra são detalhados no Gráfico 13.

Gráfico 13 - Avaliações dos Mapas Conceituais- Turma 2018 (2ª amostra)



Fonte: do autor.

Do mesmo modo que na primeira amostra, realizou-se uma primeira análise dos dados produzidos pela coleta (veja Tabela 17).

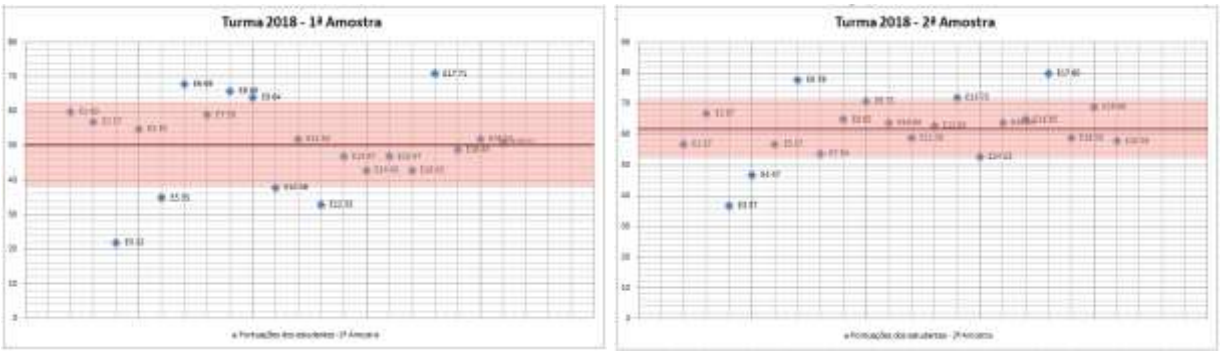
Tabela 17 - Análise das avaliações dos Mapas Conceituais- Turma 2018 (2ª amostra)

Turma 2018 (2ª amostra)		
Média da turma	61,95	
Maior nota	80,00	
Menor nota	37,00	
Desvio padrão	9,85	
Acima da média	11	55%
Abaixo da Média	9	45%
Acima da média e do desvio padrão	3	15%
Abaixo da média e do desvio padrão	2	10%

Fonte: do autor.

Ao realizar um comparativo entre a primeira e segunda coleta na turma 2018 (Tabela 18), constata-se que a média geral da turma teve um aumento (em pontos) de 50,60 para 61,95. A maior nota subiu de 71,00 pontos na primeira amostra para 80,00 na segunda coleta. Verificou-se também que a menor nota subiu de 22,00 para 37,00. Outro ponto importante para essa análise é que o desvio padrão da turma diminuiu de 12,31 para 9,85. Isso demonstra que na segunda coleta as pontuações ficaram menos dispersas em relação à média geral (Figura 18).

Figura 18 - Comparativo entre Gráficos de Dispersão entre a 1ª e 2ª Amostras – Turma 2018



Fonte: do autor.

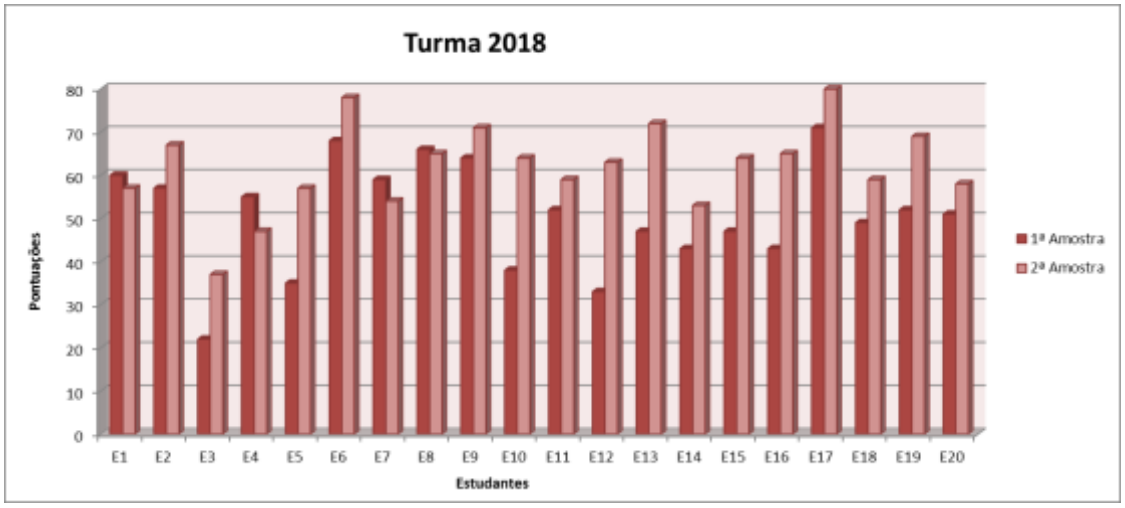
O número de estudantes acima e abaixo da média manteve-se em 55% e 45%, respectivamente, na primeira e segunda amostra.

Tabela 18 - Comparativo entre primeira X segunda amostra (Turma 2018)

Comparativo entre primeira X segunda amostra (Turma 2018)		
Item Observado	**Primeira amostra**	**Segunda amostra**
Média	50,60	61,95
Maior nota	71,00	80,00
Menor nota	22,00	37,00
Desvio padrão	12,31	9,85
Acima da média	11 (55%)	11 (55%)
Abaixo da Média:	9 (45%)	9 (45%)
Acima da média e do desvio padrão:	4 (20%)	3 (15%)
Abaixo da média e do desvio padrão:	4 (20%)	2 (10%)

A exemplo da análise realizada na turma piloto é importante também observar avanços individuais dos estudantes no processo. Tais informações são obtidas a partir do cruzamento entre os dados coletados na primeira e segunda amostra, os quais podem ser visualizados no Gráfico 14 e Tabela 19.

Gráfico 14 - Avaliações dos Mapas Conceituais- Turma 2018 (comparativo entre 1ª e 2ª amostra)



Fonte: do autor.

Desse modo, pôde se constatar que 4(quatro) estudantes (20%) apresentaram redução – caso dos estudantes E1, E4, E7, e E8 – diminuindo -5%, -15%, -8% e -2%, respectivamente. Dos demais estudantes, todos aumentaram significativamente as suas pontuações na segunda coleta, cujos percentuais podem ser observados na Tabela 19.

Tabela 19 - Comparativo individual entre 1ª e 2ª amostra (Turma 2018)

Aluno	1ª Amostra	2ª Amostra	Diferença entre 1ª e 2ª Amostra (%)
E1	60	57	-5%
E2	57	67	18%
E3	22	37	68%
E4	55	47	-15%
E5	35	57	63%
E6	68	78	15%
E7	59	54	-8%
E8	66	65	-2%
E9	64	71	11%
E10	38	64	68%
E11	52	59	13%
E12	33	63	91%
E13	47	72	53%
E14	43	53	23%
E15	47	64	36%
E16	43	65	51%
E17	71	80	13%
E18	49	59	20%
E19	52	69	33%
E20	51	58	14%

Fonte: do autor.

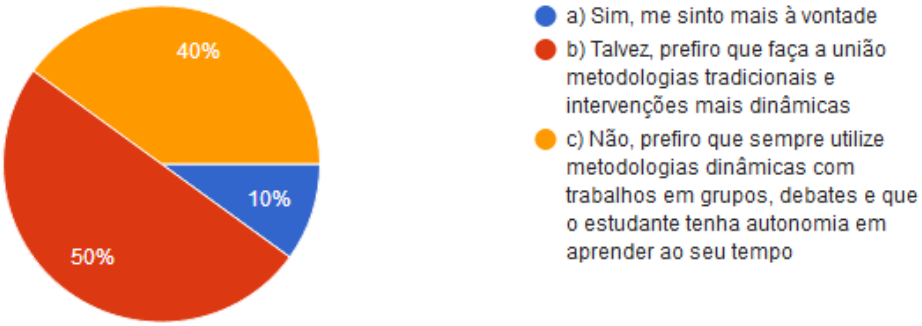
5.2.2 Questionário de opinião – Turma 2018

Depois de realizar as atividades descritas na seção 4.4.2, os estudantes responderam ao questionário de opinião sobre o uso do Greenfoot como ferramenta auxiliar no ensino de POO (APÊNDICE G). Assim, essa seção tem por objetivo realizar a tabulação e análise dos dados obtidos no questionário de opinião respondido pelos estudantes.

A primeira questão teve a intenção de mapear as preferências de dinâmicas apresentadas nas aulas, a fim de compreender o perfil da turma quanto às

preferências de aprendizagem. Nesse sentido, os estudantes foram perguntados: *Ao estudar conceitos de POO, prefere que o professor utilize uma metodologia tradicional, com o uso de apostila, slides e exercícios?* O Gráfico 15 ilustra os resultados obtidos, sendo que a grande maioria (90,0%) afirmou preferir atividades dinâmicas ou a união destas com aulas expositivas. Apenas 10% dos estudantes preferem aulas puramente expositivas.

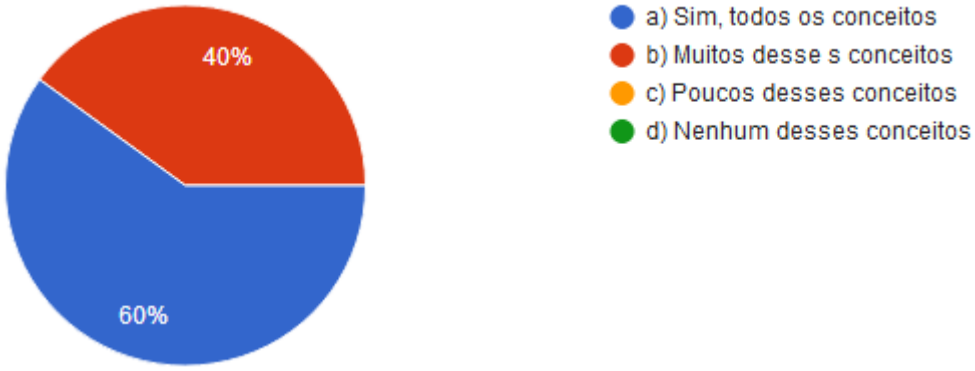
Gráfico 15 - Preferência dos estudantes quanto à metodologias empregadas pelos docentes (Turma 2018)



Fonte: do autor.

A segunda questão é relacionada às sensações da turma quanto ao ambiente Greenfoot para aprendizagem de POO. Dessa maneira, foram questionados: *Sobre a ferramenta Greenfoot, você conseguiu perceber a utilização dos principais conceitos de orientação à objetos (classe, objeto, instanciação, construtor, encapsulamento, herança, mensagem e polimorfismo)?* A partir das opiniões dos estudantes, pode-se constatar que a grande maioria deles considerou a ferramenta positiva para a aprendizagem de POO (Gráfico 16), sendo que 60,0% dos estudantes consideram que ‘todos’ os conceitos de POO podem ser visualizados a partir da utilização da ferramenta e 40,0% conseguiram visualizar ‘muitos’ conceitos de POO.

Gráfico 16 - Percepção dos estudantes da utilização dos conceitos de POO ao utilizar Greenfoot no desenvolvimento das atividades (Turma 2018).

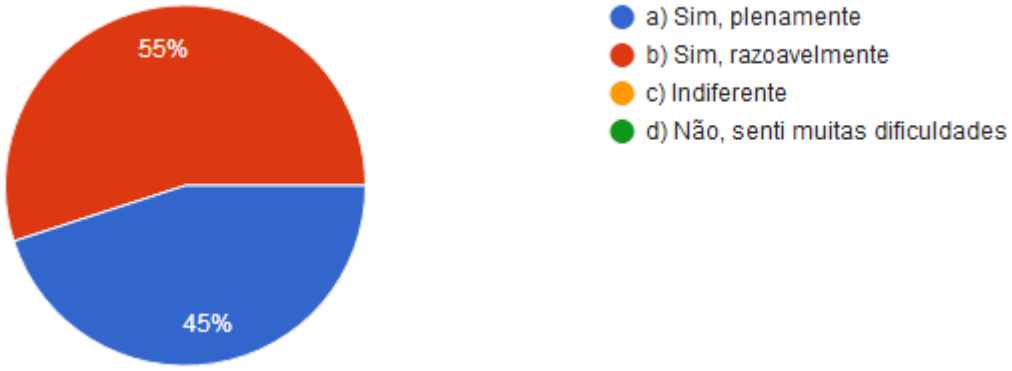


Fonte: do autor.

A terceira pergunta teve como objetivo verificar se, do ponto de vista dos estudantes, a aprendizagem dos conceitos de POO foi facilitada pelo uso do Greenfoot. Nesse caso, foram questionados: *Na sua opinião, a utilização da ferramenta Greenfoot facilitou o entendimento dos conceitos de POO?*

O Gráfico 17 demonstra que a maioria dos estudantes (55%) considerou que a partir da utilização da referida ferramenta, a aprendizagem de POO foi "razoavelmente" facilitada e 45% dos estudantes consideram que o seu uso facilita "plenamente" a aprendizagem de POO.

Gráfico 17 - Opinião dos estudantes sobre utilizar o Greenfoot para facilitar a aprendizagem de POO (Turma 2018)

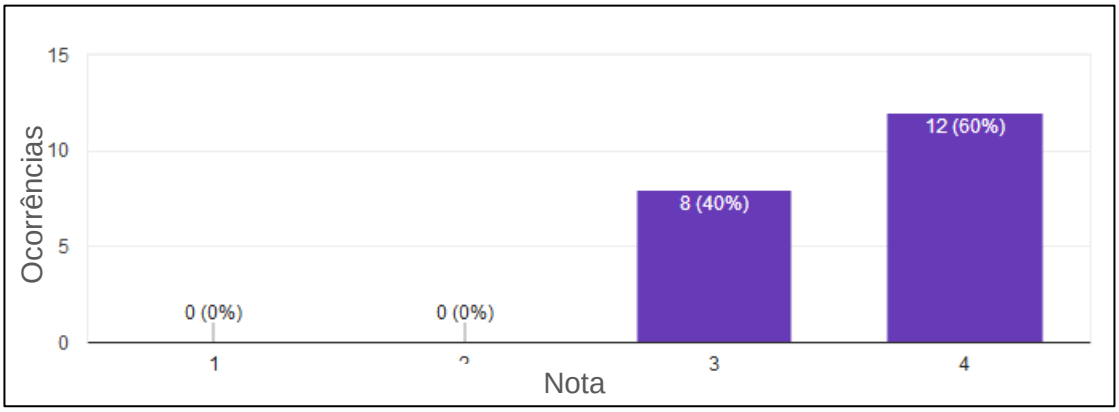


Fonte: do autor.

A quarta questão mede, em uma escala de 1 a 4, uma nota referenciada por cada estudante sobre as dinâmicas realizadas com o apoio das Unidades de estudo e Greenfoot. Nesse sentido os estudantes foram questionados: *Se você fosse atribuir*

uma nota para as atividades que envolveram o ensino de POO com o uso do Greenfoot, que nota daria?

Gráfico 18 - Notas atribuídas pelos estudantes para as atividades realizadas (Turma 2018)

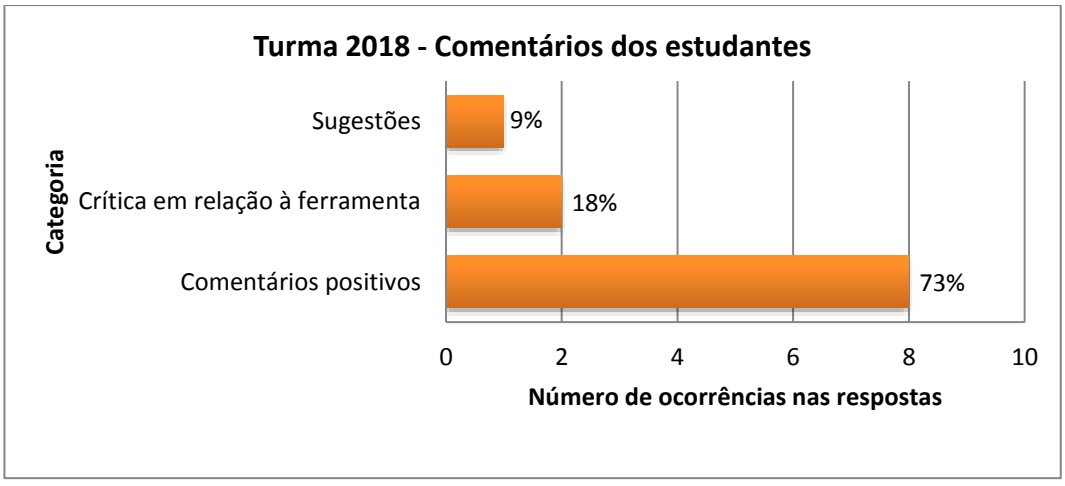


Fonte: do autor.

Pode-se perceber pelo Gráfico 18, que 60,0% dos estudantes deram a nota máxima (4) para as atividades e, 40,0% avaliaram com a segunda nota mais alta (3). Do mesmo modo que na intervenção piloto, nenhum dos estudantes apontou as notas mais baixas (1 e 2).

A última questão respondida pelos estudantes foi a de livre manifestação, e teve como objetivo colher opiniões dos estudantes quanto às atividades desenvolvidas. Para melhor analisar as respostas dos estudantes, as mesmas foram categorizadas de acordo com seu conteúdo e, nesse caso, foram organizadas nas seguintes classificações: “Sugestões”, “Críticas à ferramenta” e “Comentários positivos”. O Gráfico 19 ilustra a distribuição das respostas por categorias - sendo que em uma resposta pode haver mais de uma categoria.

Gráfico 19 - Comentários dos Estudantes (Turma 2018)



Em relação às sugestões, somente um comentário pode ser classificado nessa categoria:

> Não tornar o tempo de aprendizagem e de desenvolvimento das atividades muito acelerada, pois programação é difícil e as vezes leva tempo para compreender totalmente (E17).

A sugestão do estudante é pertinente para essa situação pois, sabidamente, o ritmo da aprendizagem é diferente de um aluno para outro.

No que se refere a críticas à ferramenta, houve dois comentários, indicando pouco material relacionado encontrado na web.

> o Greenfoot não dispõe grande quantidade de material na web (E10)!
>
> O Greenfoot não apresenta muitos conteúdos onlines disponíveis (E19).

Todavia, vale ressaltar que provavelmente os estudantes não procuraram no site oficial da ferramenta (https://www.greenfoot.org/home), indicado nas explicações iniciais. Ainda, atualmente está disponível na web, o canal do YouTube DFJUG[26] que conta com várias *playlists* de vídeos envolvendo o uso do Greenfoot, em especial, pode-se destacar a série *Greenfoot Five Minutes - Mastering the API*[27] e *GreenLabs – Laboratório de Jogos do DFJUG*[28]. A primeira série demonstra em vídeo o funcionamento e aplicação de cada classe primitiva do Greenfoot, já a segunda opção, ensina como realizar a criação de um jogo do tipo plataforma do início ao fim utilizando o Greenfoot.

Na turma de 2018, houve 8 (73%) comentários positivos em relação às atividades realizadas utilizando Greenfoot e Unidades de estudo, com destaque para:

> achei interessante, uma forma de desenvolver conhecimento através da prática (E6).
>
> As atividades foram bem legais, conseguimos fazer os desafios propostos, com muito trabalho, [...], fazendo com que fossemos mais atrás e tivemos que pesquisar mais (E8).
>
> As atividades ajudaram muito a compreender os códigos e conceitos, ainda possuo umas dúvidas sobre o funcionamento de alguns códigos, mas no demais, fácil compreensão (E2).
>
> Acredito que tanto a teoria e a prática das atividades foram muito proveitosas. Conseguimos, com este trabalho e a metodologia usada, aprender muito sobre os conceitos e como usá-los na programação (E4).

---

[26] https://www.youtube.com/channel/UC0AM5eJFl9yT4_phuzJ7J8w

[27] https://www.youtube.com/watch?v=IDElWhzh6S0&list=PLkObiPpjQQ-dboV1xELqd0SeDl8ofTfhU

[28] https://www.youtube.com/watch?v=EIlq_H3M0MA&list=PLkObiPpjQQ-fcKFelZvZjV2xMwqkRZvuL

A partir dos comentários positivos dos estudantes, pode se perceber que a grande maioria dos estudantes conseguiu compreender a importância da ferramenta e da dinâmica utilizada para ensinar e aprender os conceitos de POO. Pois nessa amostra de comentários, fica evidente que os estudantes foram estimulados a pesquisar para desenvolver as atividades, exercitando a autonomia discente na aprendizagem. De outro lado, teoria e prática estiveram em evidência na proposta apresentada, conforme se pode verificar nos comentários dos estudantes E6 e E4.

De forma a buscar formalizar matematicamente os resultados obtidos, na próxima seção é apresentado um teste estatístico para validar se a dinâmica proposta foi estatisticamente significativa no que se refere à avanços dos estudantes.

5.3 TESTE ESTATÍSTICO REALIZADO COM AS AMOSTRAS

Para averiguar se existe diferença significativa entre a avaliação do mapa conceitual antes e depois da realização das atividades descritas nas unidades de estudo, envolvendo o desenvolvimento de jogos com apoio do Greenfoot, lançou-se mão da estatística baseada em Moore (2005), no que se refere a teste de Normalidade e Teste t de *Student* (teste t), com o auxílio das ferramentas de softwares estatísticos PAST 3.20[29] e Action Stat[30] para automatizar os processos de verificação.

Inicialmente, para contribuir com a sistematização dos dados obtidos com os mapas conceituais, elaborou-se, para cada uma das amostras, um histograma, bem como um gráfico Quantil-Quantil, ou Q-Q Plot, a fim de verificar se tais dados estavam dispostos obedecendo a uma distribuição Normal. Verificar se a distribuição é Normal é importante para se definir qual teste pode ser aplicado, pois, conforme Moore (2005), caso a distribuição seja considerada Normal, pode ser aplicado o Teste t (testes paramétricos), do contrário, o Teste de Postos Sinalizados de Wilcoxon (testes não paramétricos) é indicado.

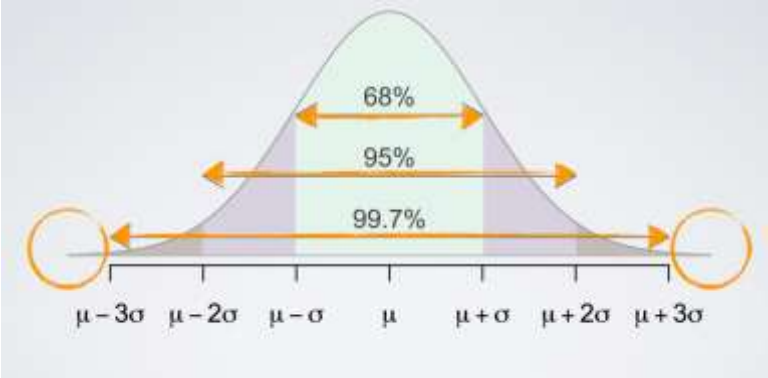
[29] O PAST 3.20 é um software gratuito para análise de dados científicos, com funções de manipulação de dados, plotagem, estatística univariada e multivariada, análise ecológica, séries temporais e análises espaciais, morfometria e estratigrafia (HAMMER, 2018b). Pode ser baixado diretamente do site oficial http://folk.uio.no/ohammer/past/.

[30] O Action Stat é um sistema estatístico desenvolvido pela Estatcamp. O início de seu desenvolvimento foi em 2005 por uma equipe de doutores em computação e estatística. O Action Stat utiliza o R, a principal linguagem de programação estatística de uso mundial (PORTAL ACTION, 2018).

Segundo Moore (2005), dados com uma distribuição Normal, apresentam curva de densidade simétrica e se enquadram na Regra 68-95-99,7. Tal regra pode ser sintetizada da seguinte maneira:

a) 68% dos valores de avaliação obtidos com os mapas conceituais devem estar no limite da sua média ± desvio padrão (σ);
b) 95% destes valores devem estar a menos de 2 desvios padrões (σ) da média;
c) 99,7% destes dados devem estar a menos de 3 desvios padrões (σ) da média.

Figura 19 - Regra 68-95-99,7 para distribuições Normais

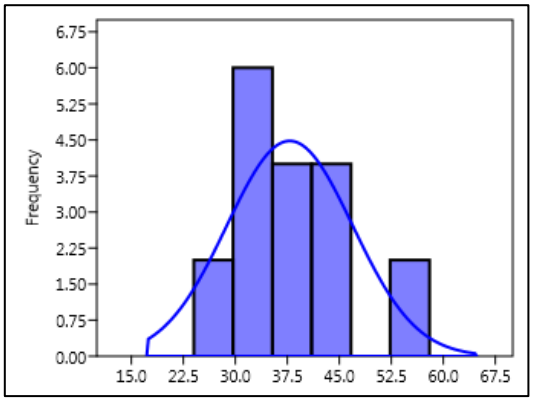


Fonte: Ponce(2018).

Assim, as pontuações obtidas pelos estudantes nas avaliações dos mapas conceituais foram submetidas à análise de normalidade pela construção de histogramas e gráficos Q-Q Plot das amostras (vide Tabela 14 e Tabela 19), utilizando-se o auxílio das ferramentas Action Stat e PAST 3.20.

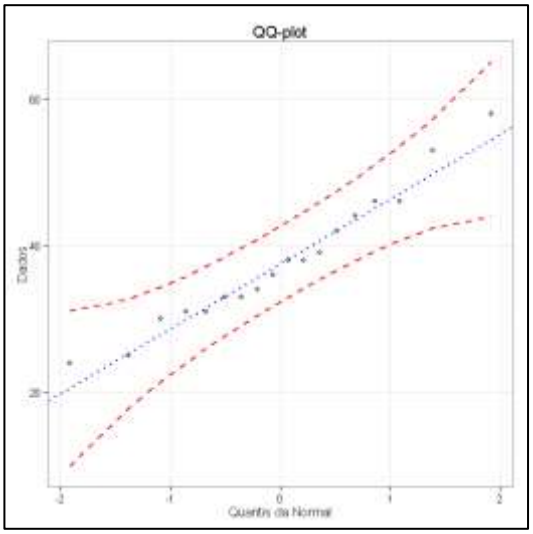
As amostras de dados da Turma Piloto geraram os resultados apresentados nos Gráficos 20, 21, 22 e 23.

Gráfico 20 - Histograma de distribuição Turma 2017 (1ª Amostra)



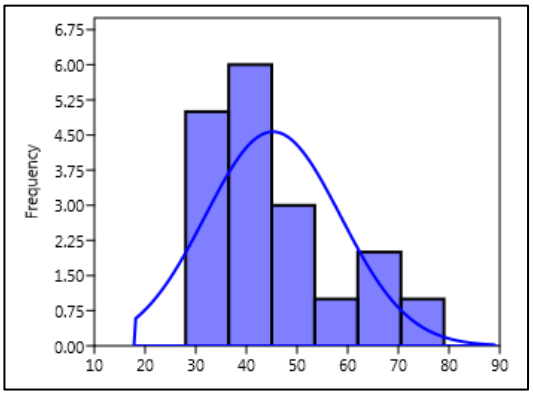
Fonte: do autor.

Gráfico 21 - Q-Q Plot de distribuição Turma 2017 (1ª Amostra)



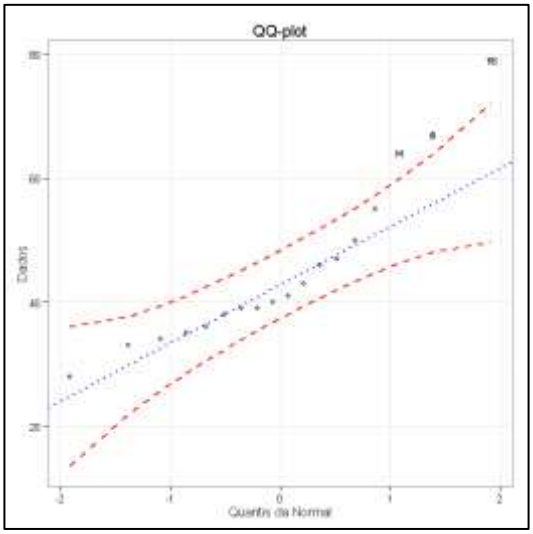
Fonte: do autor.

Gráfico 22 - Histograma de distribuição Turma 2017 (2ª Amostra)



Fonte: do autor.

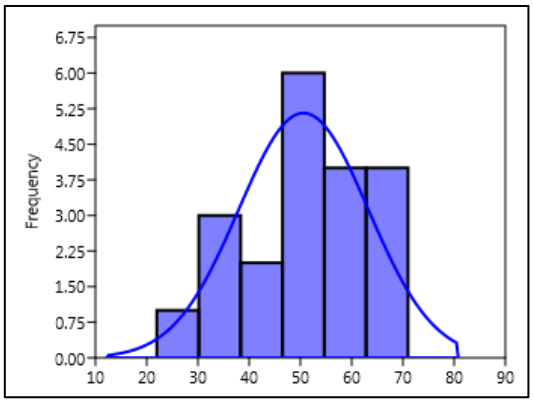
Gráfico 23 - Q-Q Plot de distribuição Turma 2017 (2ª Amostra)



Fonte: do autor.

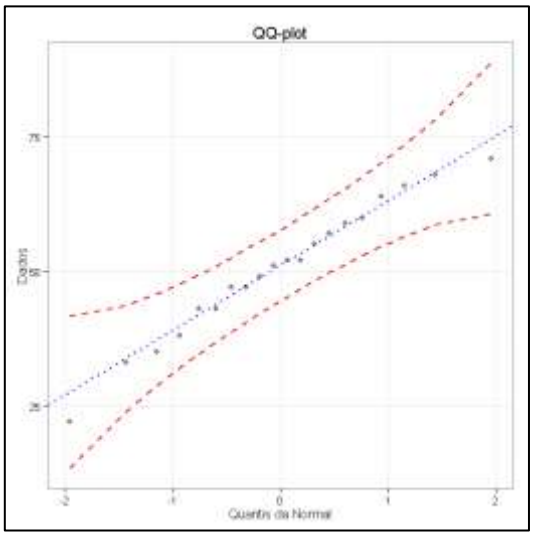
Na sequência, os dados da Turma 2018 foram submetidos à mesma análise, gerando os resultados apresentados nos gráficos 24, 25, 26 e 27.

Gráfico 24 - Histograma de distribuição Turma 2018 (1ª Amostra)



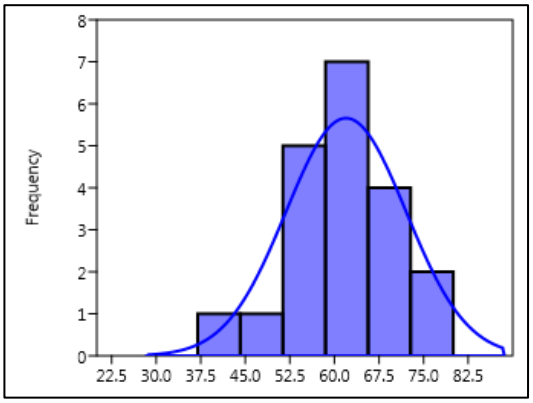
Fonte: do autor.

Gráfico 25 - Q-Q Plot de distribuição Turma 2018 (1ª Amostra)



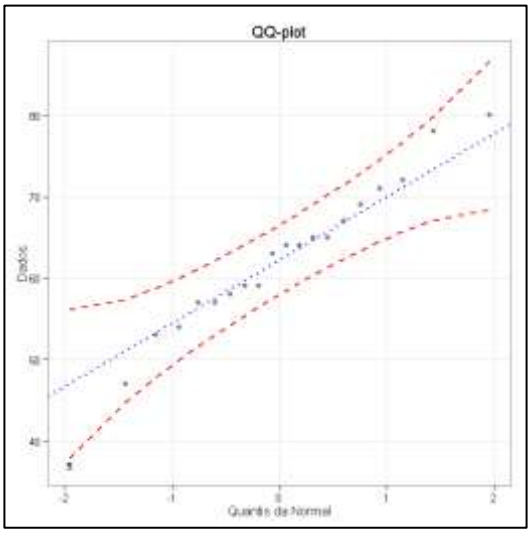
Fonte: do autor.

Gráfico 26 - Histograma de distribuição Turma 2018 (2ª Amostra)



Fonte: do autor.

Gráfico 27 - Q-Q Plot de distribuição Turma 2018 (2ª Amostra)



Fonte: do autor.

Para avaliar corretamente os histogramas, deve-se levar em consideração a regra 68-97-99,7, sendo que para que a distribuição seja Normal a curva de densidade deve ser simétrica, ter um único pico e forma de sino (MOORE, 2005). Já nos Gráficos Q-Q Plot deve-se observar:

> [...] no eixo horizontal tem-se os valores observados da variável, e no eixo vertical, os valores esperados caso a variável tenha distribuição Normal. Se há uma boa aderência dos dados à distribuição Normal os pontos estão próximos a reta de referência apresentada no gráfico (TORMAN et al, 2012, p.229).

Nesse caso, ao observar tais critérios nos histogramas e gráficos Q-Q Plot das amostras, constatou-se que na segunda amostra da turma 2017 (Gráfico 22 e Gráfico 23) existem indícios de não normalidade das distribuições, pois, o histograma da

amostra não é simétrico (possui maior volume de dados à esquerda) e gráfico Q-Q Plot possui valores distanciados da reta de referência.

Segundo Torman et al (2012), métodos gráficos têm a desvantagem de serem subjetivos, pois dependem de interpretação visual. Nesse caso, os autores sugerem que para um resultado mais objetivo, possam ser utilizados testes de aderência à distribuição Normal, tais como: Shapiro-Wilk (SW), Anderson-Darling (AD), JarqueBera (JB), entre outros. A ferramenta PAST 3.20 dispõe de tais testes entre suas opções, de maneira que as amostras foram submetidas à análise do referido software. Os resultados estão representados na Figura 20.

Figura 20 - Teste de Normalidade

	2017 (1ª Amostra)	2017 (2ª Amostra)	2018 (1ª Amostra)	2018 (2ª Amostra)
N	18	18	20	20
Shapiro-Wilk W	0,9615	0,8847	0,9804	0,9727
p(normal)	0,6314	0,03133	0,9388	0,8107
Anderson-Darling A	0,2732	0,8335	0,1385	0,2387
p(normal)	0,6235	0,02517	0,9697	0,7479
p(Monte Carlo)	0,6642	0,0246	0,9832	0,7586
Jarque-Bera JB	0,9789	4,197	0,5648	0,7411
p(normal)	0,613	0,1226	0,754	0,6903
p(Monte Carlo)	0,4103	0,0379	0,6738	0,5701

Fonte: do autor.

Para analisar tais informações, faz-se necessário observar a descrição disponível no manual de referência do PAST 3.20:

> H0: A amostra foi retirada de uma população com distribuição normal.
>
> Se o dado p (normal) for menor que 0,05, a distribuição normal pode ser rejeitada (marcada em rosa). Dos dados dos testes, o Shapiro-Wilk e Anderson-Darling são considerados os mais exatos, e os Jarque-Bera servem como referência (HAMMER, 2018, tradução nossa).

Nesse caso, a suspeita de não normalidade da segunda amostra da turma 2017, levantada pela análise empírica dos histogramas e gráficos Q-Q Plot, foi confirmada pelos testes de Normalidade realizados no PAST 3.20.

Nesse caso, segundo Moore (2005), a realização de Testes t não é recomendada, pois são sensíveis à distribuições com valores atípicos. Todavia, o autor destaca que testes t são robustos e oferecem resultados eficientes em dados dentro da normalidade e, nesse caso, antes de partir para a realização de testes não paramétricos, “podemos **transformar** nossos dados de forma que sua distribuição se aproxime mais da normal. Transformações tais como o logaritmo, que encolhem a

cauda longa de distribuições assimétricas para a direita são particularmente úteis" (MOORE, 2005, p. 539, grifo no original).

Assim, para evitar a realização de testes não paramétricos, realizou-se a transformação logarítmica dos dados das amostras da turma Piloto, conforme Figura 21:

Figura 21 - Transformação logarítmica dos valores para base 10 (log_{10})

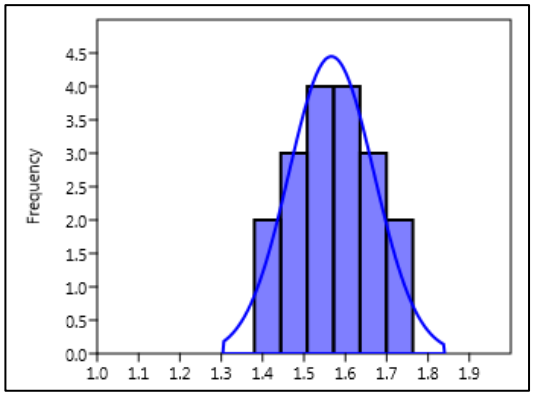
Turma 2017(Piloto)		
Aluno	1ª Amostra	2ª Amostra
A2	30	36
A3	33	55
A4	34	38
A6	46	50
A7	33	33
A9	58	67
A11	25	35
A12	24	46
A14	36	28
A15	38	41
A16	53	64
A17	31	34
A18	42	43
A19	38	47
A20	44	39
A23	31	39
A24	39	40
A26	46	79

Turma 2017(Piloto)		
Aluno	1ª Amostra	2ª Amostra
A2	1,47712125471966	1,55630250076729
A3	1,51851393987789	1,74036268949424
A4	1,53147891704226	1,57978359661681
A6	1,66275783168157	1,69897000433602
A7	1,51851393987789	1,51851393987789
A9	1,76342799356294	1,82607480270083
A11	1,39794000867204	1,54406804435028
A12	1,38021124171161	1,66275783168157
A14	1,55630250076729	1,44715803134222
A15	1,57978359661681	1,61278385671974
A16	1,72427586960079	1,80617997398389
A17	1,49136169383427	1,53147891704226
A18	1,62324929039790	1,63346845557959
A19	1,57978359661681	1,67209785793572
A20	1,64345267648619	1,59106460702650
A23	1,49136169383427	1,59106460702650
A24	1,59106460702650	1,60205999132796
A26	1,66275783168157	1,89762709129044

Fonte: do autor.

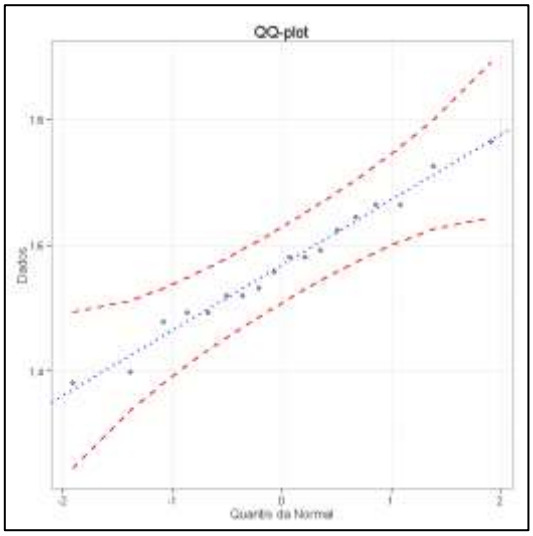
Na sequência, foram realizados novos testes de normalidade para verificar se, a partir da transformação, os dados passaram a se aproximar de uma distribuição Normal (Gráficos 28, 29, 30 e 31).

Gráfico 28 - Histograma de distribuição Turma 2017 (1ª Amostra) – com dados transformados



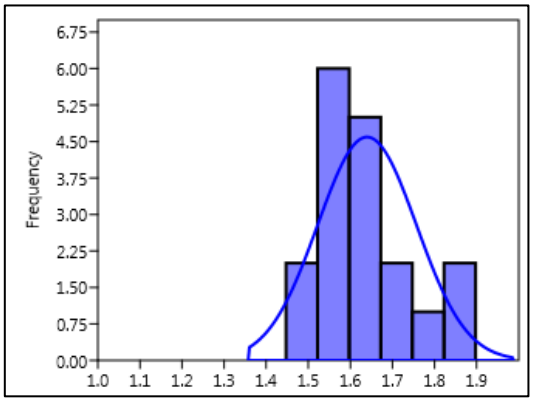
Fonte: do autor.

Gráfico 29 - Q-Q Plot de distribuição Turma 2017 (1ª Amostra) com dados transformados



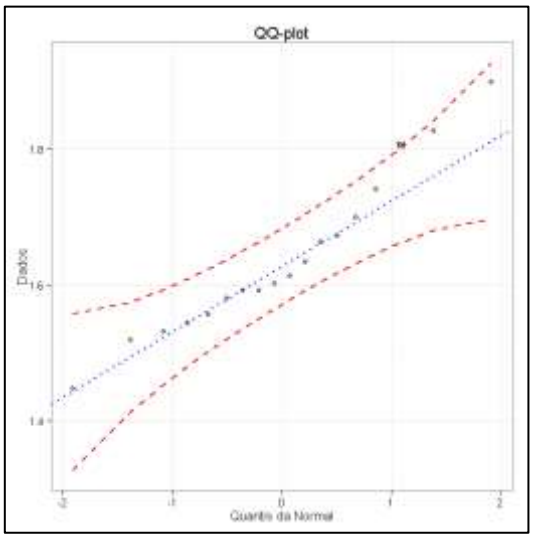
Fonte: do autor.

Gráfico 30 - Histograma de distribuição Turma 2017 (2ª Amostra) – com dados transformados



Fonte: do autor.

Gráfico 31 - Q-Q Plot de distribuição Turma 2017 (2ª Amostra) com dados transformados



Fonte: do autor.

Pode-se verificar pelo histograma e gráfico Q-Q Plot que a segunda amostra da turma 2017, mesmo depois da transformação, ainda apresenta uma leve distorção na distribuição Normal, mas os testes de normalidade Shapiro-Wilk (SW), Anderson-Darling (AD) e JarqueBera (JB), comprovam que há normalidade nos dados, ou seja, p(normal) >0,05, conforme apresentado na Figura 22.

Figura 22 - Teste de Normalidade Turma 2017 com dados transformados

	2017 (1ª Amostra)	2017 (2ª Amostra)
N	18	18
Shapiro-Wilk W	0,9816	0,9508
p(normal)	0,9653	0,4384
Anderson-Darling A	0,155	0,4109
p(normal)	0,9454	0,3065
p(Monte Carlo)	0,9649	0,316
Jarque-Bera JB	0,2136	1,325
p(normal)	0,8987	0,5154
p(Monte Carlo)	0,8933	0,2629

Fonte: do autor.

Diante de tais resultados, pôde-se realizar o teste t a fim de verificar se houve avanço das médias dos estudantes na segunda amostra. Nesse caso, utilizou-se o software Action Stat para realizar a testagem de maneira automatizada. Porém, primeiramente, antes de se realizar tais testes, as hipóteses nula (H_0) e alternativa (H_a), foram formuladas:

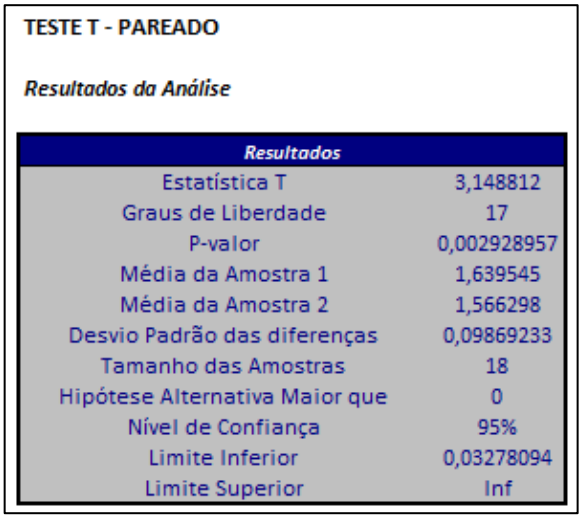
H_0: $\mu \leq \mu_0$

H_a: $\mu > \mu_0$

A hipótese nula coincide com a seguinte afirmação: a média da segunda amostra (μ) é menor ou igual a média da primeira amostra (μ_0). Já a hipótese alternativa (H_a) afirma que: média da segunda amostra (μ_2) é maior que a média da primeira amostra (μ_1).

Os resultados obtidos pelo teste t das turmas estão ilustrados nas Figuras 21 e 22:

Figura 23 - Resultado teste t Turma 2017



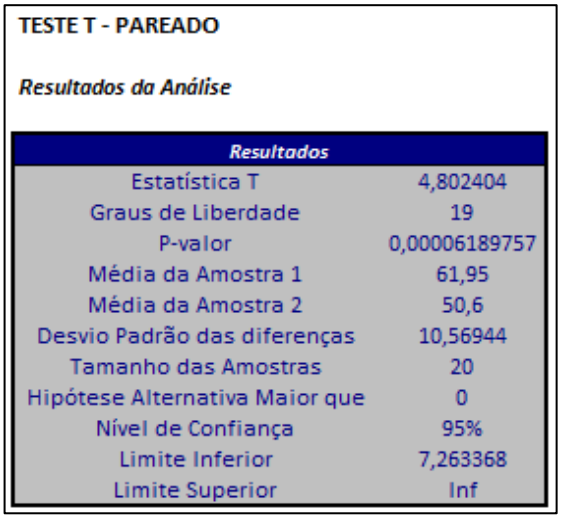
TESTE T - PAREADO

Resultados da Análise

Resultados	
Estatística T	3,148812
Graus de Liberdade	17
P-valor	0,002928957
Média da Amostra 1	1,639545
Média da Amostra 2	1,566298
Desvio Padrão das diferenças	0,09869233
Tamanho das Amostras	18
Hipótese Alternativa Maior que	0
Nível de Confiança	95%
Limite Inferior	0,03278094
Limite Superior	Inf

Fonte: do autor.

Figura 24 - Resultado teste t Turma 2018



TESTE T - PAREADO

Resultados da Análise

Resultados	
Estatística T	4,802404
Graus de Liberdade	19
P-valor	0,00006189757
Média da Amostra 1	61,95
Média da Amostra 2	50,6
Desvio Padrão das diferenças	10,56944
Tamanho das Amostras	20
Hipótese Alternativa Maior que	0
Nível de Confiança	95%
Limite Inferior	7,263368
Limite Superior	Inf

Fonte: do autor.

Para analisar os resultados do teste, faz-se necessário recorrer a Moore (2005) e observar as informações retornadas pelo teste. Ao testar hipóteses, a hipótese nula (H_0) enuncia a afirmação contra a qual procuramos evidência e as dinâmicas adotadas pelos testes buscam provar a H_0, assim:

> A probabilidade, calculada supondo-se que H_0 é verdadeira, de que a estatística de teste assumisse um valor tão ou mais extremo do que o valor realmente observado é chamada de **P-valor**. Quanto menor o P-valor, mais forte a evidência contrária a H_0 fornecida pelos dados (MOORE, 2005, p. 296, grifo no original).

Nesse caso, para obtermos a conclusão do teste, basta observar o quantitativo retornado para P-valor e, caso seja menor que o nível de significância – que nesse caso é 0,05 (5%) – rejeita-se a hipótese nula. Tendo em vista que os *P-valores* obtidos no teste foram *0,002928957* e *0,00006189757,* **conclui-se que as evidências apontam para a aceitação de que a hipótese alternativa (H_a) é verdadeira.**

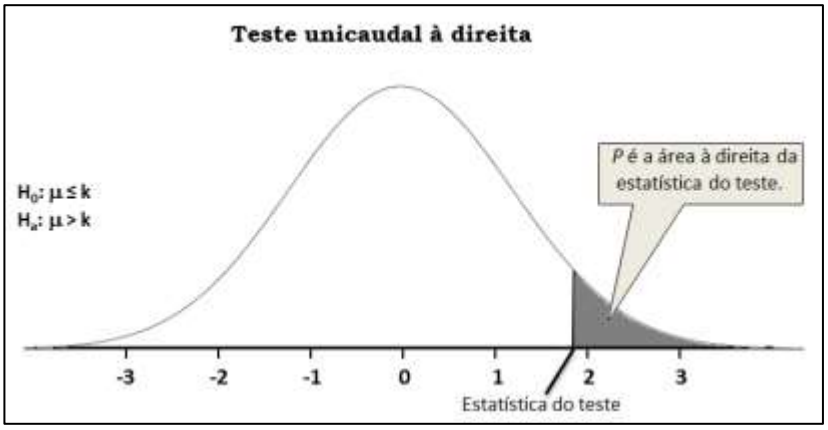
Para ilustrar em gráficos o resultado, realizou-se um *Teste unicaudal à direita*[31] (Gráfico 32), sendo que, nesse caso, se o valor da *Estatística T* for maior que o Tc[32], rejeita-se H_0, caso contrário, não se pode rejeitar a hipótese nula.

---

[31] Segundo Larson e Farber (2010, p.298) "se a hipótese alternativa H_a contém um símbolo maior que igualdade (>), o teste de hipótese é um **teste unicaudal à direita**" (grifo no original). No caso em tela, a hipótese alternativa contém o símbolo maior que a igualdade (Ha: $\mu > \mu 0$), nesse caso realizou-se o teste unicaudal à direita.

[32] Tc: é o valor crítico de t, obtido pelo cruzamento dos graus de liberdade (gl) do teste e o nível de confiança disponível na Tabela de "Valores críticos da distribuição t" – Anexo I.

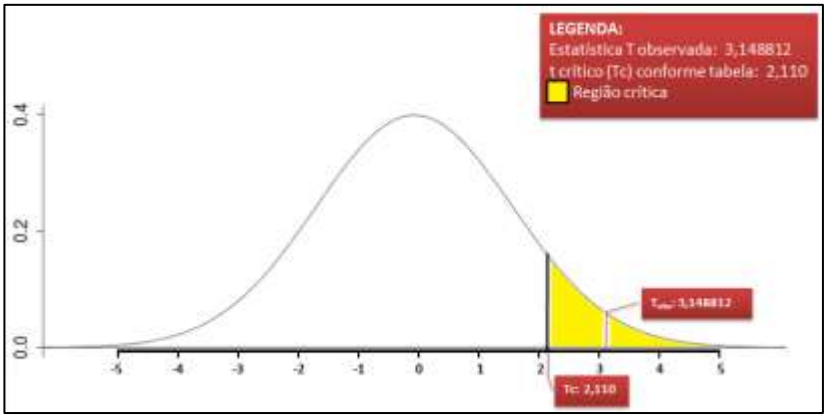
Gráfico 32 - Representação do gráfico de teste unicaudal à direita



Fonte: Larson e Farber (2010, com adaptações).

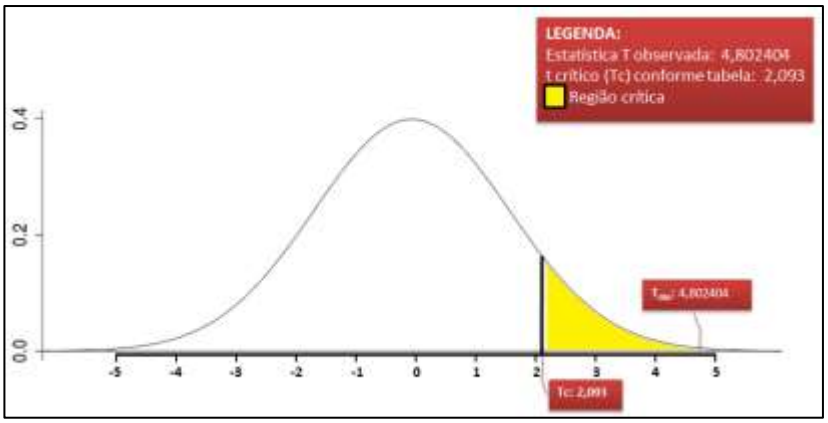
Os resultados estão apresentados nos gráficos 33 e 34.

Gráfico 33 - Teste t Turma 2017



Fonte: do autor.

Gráfico 34 - Teste t Turma 2018



Fonte: do autor.

Em ambos os casos o valor da Estatística T_{obs} está dentro da região crítica e, nesse caso, rejeita-se a hipótese nula aceitando a hipótese alternativa. Nesse caso a

conclusão do teste t a que os dados foram submetidos é de que **ao nível de significância de 5%, há evidências de que a média da segunda amostra seja maior que a média da primeira**.

Por fim, diante de todos os resultados obtidos na análise estatística realizada nessa seção - pelo teste t e representações gráficas – constata-se que houve melhoria nas médias dos estudantes, comparando-se a média obtida antes e depois das atividades organizadas pelas unidades de estudo com apoio do Greenfoot para o ensino e aprendizagem de POO.

5.4 AVALIAÇÃO DO PROCESSO

Esta seção tem o objetivo de realizar uma análise crítico-reflexiva acerca das intervenções pedagógicas realizadas com as turmas piloto e 2018, bem como analisar os materiais produzidos e utilizados nas unidades de estudo. Esse momento de análise é característico da atividade profissional docente, fomentado pela necessidade de compreender e aprimorar o seu papel formativo, mediante a práxis pedagógica.

Tal análise trata-se de um movimento fundamental para atuação docente, pois, conforme Freire (1996, p. 42-43), “a prática docente crítica, implicante do pensar certo, envolve o movimento dinâmico, dialético, entre o fazer e o pensar sobre o fazer”. Nesse caso, para além da necessidade de se planejar cuidadosamente cada intervenção realizada com determinado grupo de estudantes, faz-se também necessário refletir sobre “o fazer”. A atitude do profissional docente em pensar criticamente a prática realizada é o fundamento que permite melhorar atuações futuras (FREIRE, 1996).

Desse modo, é importante analisar as atividades desenvolvidas com as turmas e verificar aspectos positivos e o que necessita ser melhorado. Nesse sentido, as repostas dos estudantes no questionário de opinião, juntamente com as médias calculadas pela análise dos mapas conceituais, oferecem bom referencial para análise do material produzido e utilizado nas aulas. Todavia, é fundamental também explicitar as impressões do pesquisador acerca do processo, visto que se trata de uma pesquisa-ação e, nesse caso, pesquisador também assume o papel de pesquisado ao estar imerso no contexto da pesquisa (THIOLLENT, 2011; BARBIER, 2007).

Assim, no que se refere à turma piloto (2017), foram desenvolvidas e utilizadas três unidades de estudos (Apêndices A, B e C), totalizando 13(treze) horas de atividades. A receptividade foi positiva, os grupos se formaram rapidamente e, assim que foram passadas as primeiras orientações, os estudantes se empenharam na resolução das atividades propostas.

Inicialmente, a Unidade de Estudo 1.1 (APÊNDICE A), que consiste em visualizar o vídeo e abstrair os objetos nele verificados – bem como suas características e funcionalidades - foi de fácil resolução para os estudantes. Estes já estavam familiarizados com dinâmicas parecidas, exceto pela utilização do vídeo que, nesse caso, tem a finalidade de produzir nos estudantes sensações que aproximam o objeto de estudo a sua realidade. Nesse processo, percebe-se parte do Ciclo de Aprendizagem proposto por Kolb(1984) está presente, pois os estudantes puderam vivenciar uma "experiência concreta" e, a partir disso, pensar acerca do que visualizaram ("observação reflexiva") e formular hipóteses ("conceitualização abstrata").

A próxima atividade, definida pela Unidade de Estudo 1.2 (APÊNDICE B), também composta por exemplos e vídeo, foi incrementada com a disponibilização de uma modelagem pré-definida em que os estudantes tinham a tarefa de programar e, dessa forma, utilizar os conceitos de POO envolvidos no processo. Nessa unidade de estudo, o ciclo de aprendizagem iniciado na unidade anterior (1.1) se completa, pois, os estudantes têm a tarefa de programar as hipóteses anteriormente levantadas, o que gera uma "experimentação ativa" (KOLB, 1984).

Embora lhes fosse disponibilizado material instrucional (Unidade de Estudo 1.2), verificou-se que poucos estudantes leram seu conteúdo na íntegra, pois, muitas dúvidas surgiram e, nesse momento, foi fundamental o papel do professor mediador, no sentido de sanar os questionamentos dos estudantes por meio de contextualização teórica envolvida. Tal situação corrobora com os argumentos de Veen e Wrakking (2009), pois estudantes da atualidade pouco se preocupam em "ler o manual", buscam aprender pela experimentação e pelos auxílios da sua rede de contatos.

As atividades da Unidade de Estudo 1.3 (APÊNDICE C) englobam a maioria dos conceitos de POO trabalhados nas aulas. A realização dessas atividades gerou muito trabalho e empenho dos estudantes para concluí-las dentro do prazo estipulado (6h). Em geral, a partir dos auxílios e da colaboração entre os pares, todos os estudantes conseguiram completar a tarefa que, ao final, possibilitou a realização

completa do Ciclo de Aprendizagem de Kolb, pois, os estudantes assistiram ao vídeo explicativo (“experiência concreta”), analisaram e formularam suas hipóteses de solução a partir da programação (“observação reflexiva” e “conceitualização abstrata”). Finalmente, testaram o código por eles desenvolvido (“experimentação ativa”), o que lhes proporcionou nova “experiência concreta”, recomeçando o ciclo.

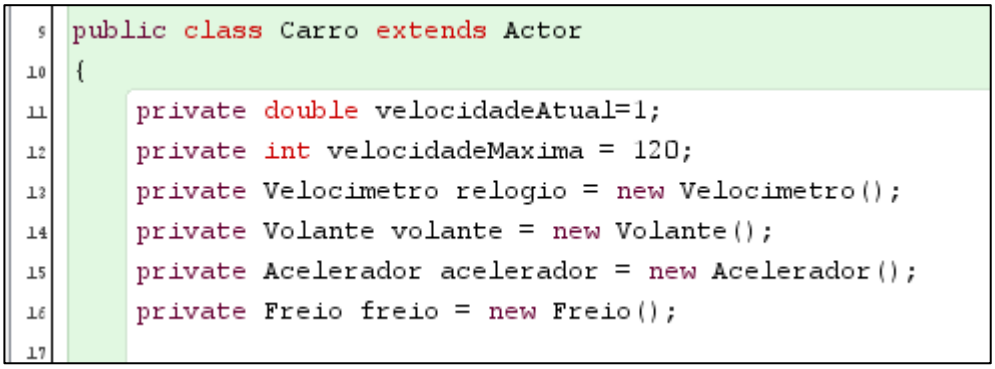
As atividades desenvolvidas com a turma piloto foram sequenciais, embora possam ser abordadas de maneira independente, a Unidade de Estudo 1.1 (APÊNDICE A) dá suporte à Unidade de Estudo 1.2 (APÊNDICE B) e, esta, para a Unidade de Estudo 1.3 (APÊNDICE C). Assim, a Unidade de Estudo 1.3 agrupa todos os conceitos trabalhados nas unidades de estudo anteriores.

Os conteúdos abordados nas unidades de estudo são fundamentais para que os estudantes compreendam a essência do paradigma de programação orientada a objetos – conceito de abstração está evidente na modelagem UML que desempenha papel fundamental na proposta, pois, construir o protótipo do jogo de maneira adequada depende do planejamento prévio e respeito criterioso à modelagem definida (SANTOS, 2003).

Demais conceitos como classe, objeto, instanciação e herança são naturalmente utilizados na IDE Greenfoot pois, objetos inseridos no cenário precisam, primeiramente, ser descritos em classes (atributos e métodos) e instanciados. Ainda, o conceito de herança está implícito em praticamente todas as classes, pois quase que a totalidade delas deriva (*extends*) de *Actor* e *World* (nativas da ferramenta) e, nesse caso, já herdam atributos e métodos da sua classe ancestral.

Além disso, a Unidade de Estudo 1.3 (APÊNDICE C), em especial, agrega muitos elementos significativos para que os estudantes compreendam os conceitos de POO, pois, por exemplo, ao criar a classe “Carro” contendo atributos que não são tipos primitivos de dados, como: “Volante”, “Velocímetro”, “Freio” e “Acelerador” (Figura 25), rompe-se a ideia de que variáveis são sempre de tipos primitivos, pois estão declarando (instanciando) variáveis complexas – não primitivas como estavam acostumados (int, float, double, etc) – que no caso da POO não são variáveis, são instâncias, e que por esse motivo carregam além de valores a possibilidade de executar ações, pois esse é o diferencial da POO em relação às demais técnicas de desenvolvimento (TUCKER e NOONAM, 2010).

Figura 25 - Fragmento de código da classe Carro (Projeto Simulador)

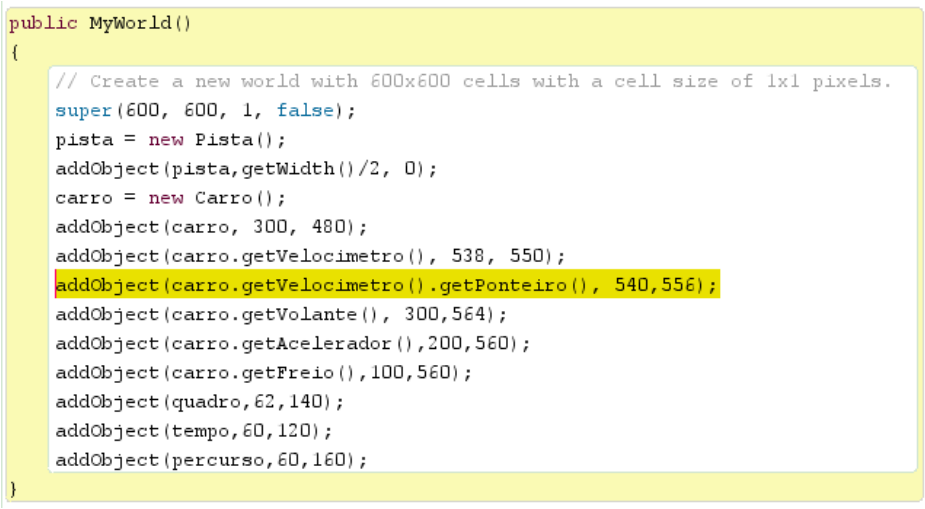


```
9  public class Carro extends Actor
10 {
11     private double velocidadeAtual=1;
12     private int velocidadeMaxima = 120;
13     private Velocimetro relogio = new Velocimetro();
14     private Volante volante = new Volante();
15     private Acelerador acelerador = new Acelerador();
16     private Freio freio = new Freio();
17
```

Fonte: do autor.

Ainda nesse viés, foi importante também oportunizar para os estudantes a percepção de que objetos podem ser atributos de outras classes, o que torna esse paradigma de programação especialmente valioso para o desenvolvimento de software (SANTOS, 2003). Essa situação pôde ser verificada pelo menos na relação existente entre carro, velocímetro e ponteiro, pois, o ‘carro’ possui o atributo ‘velocímetro’ que por sua vez contém o atributo ‘ponteiro’, assim, a partir do contexto do cenário principal do jogo (*MyWorld*), para acessar o objeto ‘ponteiro’, é necessário fazê-lo por intermédio do carro e, a partir dele acessar o velocímetro, para daí então alcançar o ponteiro (Figura 26).

Figura 26 - Construtor da classe MyWorld (Projeto Simulador)



```
public MyWorld()
{
    // Create a new world with 600x600 cells with a cell size of 1x1 pixels.
    super(600, 600, 1, false);
    pista = new Pista();
    addObject(pista,getWidth()/2, 0);
    carro = new Carro();
    addObject(carro, 300, 480);
    addObject(carro.getVelocimetro(), 538, 550);
    addObject(carro.getVelocimetro().getPonteiro(), 540,556);
    addObject(carro.getVolante(), 300,564);
    addObject(carro.getAcelerador(),200,560);
    addObject(carro.getFreio(),100,560);
    addObject(quadro,62,140);
    addObject(tempo,60,120);
    addObject(percurso,60,160);
}
```

Fonte: do autor.

As atividades desenvolvidas com a turma piloto, embora tenham sido positivas, apresentaram problemas de ordem prática no contexto do jogo proposto, pois o desafio consistiu no desenvolvimento de um simulador de um automóvel em que possibilita ao jogador acelerar, frear e mudar de direção, porém, não há um objetivo a

ser alcançado que indicasse vitória ou algum desafio/conflito que possibilitasse derrota (PRENSKY, 2012). Tal situação foi apontada nos comentários dos estudantes, em especial pelo aluno A25, ao comentar que “faltou objetivo no jogo”, o que expõe essa fragilidade na atividade a eles apresentada.

Assim, a fim de contornar essa situação, para a segunda intervenção, tais aspectos foram considerados nas propostas apresentadas aos estudantes, sobretudo no que se refere ao projeto “Simulador”, o qual foi ajustado (APÊNDICE E), pela incorporação de contador de tempo, distância e de obstáculos que surgem aleatoriamente no cenário (Figura 27). Nesse sentido, o jogador tem o desafio de desviar dos referidos obstáculos, pois, em caso de colisão o jogador perde – a vitória acontece ao percorrer cinco mil metros sem colidir.

Figura 27 - Cena capturada do Projeto Simulador em execução

Fonte: do autor.

Outro fator a ser considerado em relação à intervenção piloto, está relacionado ao comentário do estudante A18, que sugere “desafiar a criação de jogos sozinho com o uso do Greenfoot”. Nesse caso, é possível pensar um cenário sugerido pelo referido estudante, desde que o professor estabeleça critérios mínimos a serem contemplados pelo jogo a ser criado. Todavia, é importante ressaltar que uma atividade com essa dinâmica demanda maior empenho do professor nos auxílios e resolução dos problemas – pois, possivelmente, devem surgir múltiplas situações não planejadas

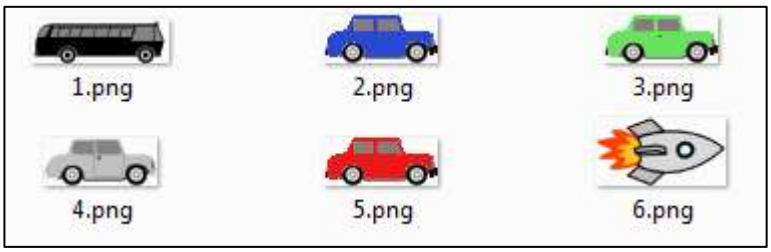
pelo docente. Embora estimular a criatividade e a autonomia dos estudantes seja potencialmente valioso, no caso de turmas maiores, pode-se considerar a inviabilidade dessa proposta. Sobretudo, é importante frisar que isso depende, fundamentalmente, do perfil da turma e do nível de dificuldade proposto.

Em relação à segunda intervenção (turma 2018) foram utilizadas duas unidades de estudos (Apêndices D e F), totalizando 15(quinze) horas de atividades. Do mesmo modo que aconteceu com a turma piloto, a receptividade foi boa, os estudantes estiveram atentos às instruções iniciais, formaram grupos e se empenharam na resolução das tarefas.

Inicialmente, na Unidade de Estudo 2.1 (APÊNDICE D), os estudantes conseguiram solucionar a tarefa sem maiores problemas, visto que o material disponibilizado estava explicado em detalhes e os auxílios do mediador proporcionaram certa segurança no desenvolvimento da proposta. A referida unidade de estudo tem como objetivo abordar os conceitos elementares da POO e, para isso, sugere a construção de um protótipo chamado “Travessia”, onde o ator principal necessita atravessar uma movimentada avenida para chegar ao supermercado.

Nessa unidade de estudo, o conceito de atributo é alcançado na classe “Carro”, ao definir uma ‘velocidade’ e ‘imagem’ individual para cada objeto criado (Figura 28). Nesse aspecto, ao criar os objetos e inseri-los no cenário, aborda-se também o conceito de instanciação. Definir tais características depende de descrevê-las na classe e, dessa forma tal conceito está constantemente presente no processo, pois cada objeto inserido no contexto do jogo tem origem em uma classe.

Figura 28 - Imagens possíveis para objetos da classe Carro (Projeto Travessia)



Fonte: do autor.

Cada “carro” inserido no cenário tem valores aleatórios no que se refere à velocidade e imagem, e isso é realizado dentro do construtor (Figura 29), evidenciando também esse conceito.

Figura 29 - Construtor da classe Carro (Projeto Travessia)

```
public Carro(){
    int x = 1+Greenfoot.getRandomNumber(6);//gerando numero aleatório de 1 a 6
    this.setImage(x+".png");//definindo a imagem
    this.velocidade = x*2;//definindo a velocidade
}
```

Fonte: do autor.

O conceito de método é abordado em praticamente todas as classes, porém, isso fica muito evidente ao criar a classe ‘Homem’, pois, tem movimentos à acima, abaixo, para a esquerda e direita. Tal conceito é visualizado na prática e os estudantes podem ter as sensações visuais em reflexo do que fora desenvolvido em nível de programação.

O conceito de mensagem é permanentemente testado, pois a colisão entre objetos tem reflexos no comportamento dos envolvidos. Por exemplo, caso o objeto do tipo ‘Homem’, colida com o muro (Figura 30), sua posição é alterada e caso colida com um veículo, o jogo é encerrado (Figura 31).

Figura 30 - Método colideNoMuro (Projeto Travessia)

```
public void colideNoMuro(){
   Actor a = getOneIntersectingObject(Muro.class);//captura objeto Muro caso o Homem
                                                  // entre nas mesmas coordenadas
    if(a!=null){
    if(a.getY()<this.getY()){
        setLocation(getX(), a.getY()+40);//altera a posição para baixo
    }else{
        setLocation(getX(), a.getY()-40);//altera a posição para cima
    }
}
}
```

Fonte: do autor.

Figura 31 - Método perde classe Homem (Projeto Travessia)

```
public void perde(){
    Actor a =getOneIntersectingObject(Carro.class);//colide com carro
    if(a != null){
        this.getWorld().addObject(new GameOver(), 500, 250);//mensagem de 'Game Over'
        Greenfoot.stop();//pára o jogo
    }
}
```

Fonte: do autor.

Na unidade de estudo 2.3 (APÊNDICE F) trabalhada com os estudantes na segunda intervenção, todos os conceitos de POO são abordados. A proposta do jogo

consiste em um ator principal (herói) representado pela imagem de uma joaninha, que necessita colher as maçãs dispostas aleatoriamente no cenário. Cada maçã colhida incrementa a pontuação do ator principal e ele vence ao somar 10(dez) pontos. Como desafio, o 'herói' necessita desviar-se de asteroides que caem aleatoriamente, pois caso colida, o jogador perde e o jogo é encerrado. Além disso, o jogo apresenta bônus (SuperMacas) representado por maçãs verdes que passam aleatoriamente no cenário e caso consiga captar uma delas o herói é promovido ao status de SuperHeroi.

É possível perceber nesse protótipo que, de certa forma, todos os elementos estruturais do jogo, definidos por Prensky (2012), estão presentes: *regras; metas* ou *objetivos*; *resultados* e *feedback*; *conflito/competição/desafio/oposição*; *interação* e *representação* ou *enredo*. Tais elementos proporcionam significado à tarefa e estimulam os estudantes a se envolver na proposta.

No que toca aos conceitos de POO imbricados, a Unidade de Estudo 2.3 (APÊNDICE F) contempla todos os elementos abordados nas aulas, pois os conceitos de classe, objeto, atributo, método, instanciação e mensagem são da mesma forma que na unidade de estudo 2.1, intrínsecos no processo. Todavia, a ênfase dada nessa atividade é relacionada à encapsulamento, herança e polimorfismo.

O encapsulamento que tem como finalidade ocultar a complexidade da implementação, está em evidência nessa atividade, pelo bloqueio do acesso direto à atributos – materializado pela definição do padrão de acesso privado (*private*) (Figura 32) e pela implementação e utilização de métodos assessores que tornam a sua utilização transparente ao desenvolvedor (Figura 33).

Figura 32 - Atributos privados na classe Heroi (Projeto Colheita das Maçãs)

```
16      private int pontos;
17      private boolean colheuSuperMaca=false;
```

Fonte: do autor.

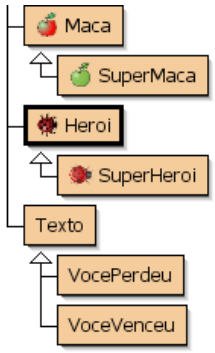
Figura 33 - Métodos acessores para atributos da classe Heroi (Projeto Colheita das Maçãs)

```
public int getPontos(){
    return this.pontos;
}

public void setPontos(int pontos){
    this.pontos=pontos;
}

public void setColheuSuperMaca(boolean valor){
    colheuSuperMaca = valor;
}

public boolean getColheuSuperMaca(){
    return  colheuSuperMaca ;
}
```

O conceito de herança esteve presente na criação de qualquer uma das classes de atores do cenário, pois derivam impreterivelmente da classe Actor, nativa do Greenfoot. O mesmo acontece com a classe ‘Cenário’ que deriva de ‘World’, também nativa da ferramenta.

Além disso, mais diretamente com a modelagem do protótipo, a especialização de classes esteve presente também em ‘SuperMaca’, SuperHeroi, ‘VocePerdeu’ e ‘VoceVenceu’, tendo em vista que são subclasses de, respectivamente, ‘Maca’, ‘Heroi’ e ‘Texto’ (Figura 34).

Figura 34 - Hierarquia de classes (Projeto Colheita das Maçãs)



Fonte: do autor.

Todavia, o principal benefício da herança está na possibilidade de reaproveitar funcionalidades herdadas e, sobretudo, na possibilidade de reimplementá-las caracterizando diversas formas (polimorfismo)(SANTOS, 2003; JOYANES AGUILAR, 2008; TUCKER e NOOAM, 2010). Essa relação é verificada em especial entre as classes ‘Heroi’ e ‘SuperHeroi’, no que se refere ao método ‘pontua()’. Na classe ‘Heroi’,

os objetos pontuam uma unidade a cada maçã colhida (Figura 35) e, na classe SuperHerói, os objetos pontuam duas unidades ao colher uma maçã (Figura 36).

Figura 35 - Método 'pontua()' da classe Heroi (Projeto Colheita das Maçãs)

```
public void pontua(){
    Actor a = getOneIntersectingObject(Maca.class);
    if(a != null){
        this.pontos++;
        Greenfoot.playSound("slurp.wav");
    }
}
```

Fonte: do autor.

Figura 36 - Método 'pontua()' da classe SuperHeroi (Projeto Colheita das Maçãs)

```
public void pontua(){
    Actor a = getOneIntersectingObject(Maca.class);
    if(a != null){
        this.setPontos(this.getPontos()+2);
        Greenfoot.playSound("slurp.wav");
    }
}
```

Fonte: do autor.

O ápice da utilização do conceito de herança e polimorfismo, nesse projeto ocorre quando o ‘heroi’ consegue colher uma supermaçã, pois nesse momento ele passa ao status de ‘SuperHeroi’ e tem comportamentos alterados. Essa ação é realizada pelo método ‘atualizaHeroi()’ dentro da classe ‘Cenário’ (Figura 37).

Figura 37 - Método 'atualizaHeroi()’ classe Cenário (Projeto Colheita das Maçãs)

```
public void atualizaHeroi(){
    if(heroi.getColheuSuperMaca()&& (heroi.getClass().getName().equals("Heroi"))){
        int x=heroi.getX();//salvando valor da coordenada x
        int y=heroi.getY();//salvando valor da coordenada y
        int p=heroi.getPontos();//salvando a pontuação atual
        int r=heroi.getRotation();//salvando a rotação do ator
        this.removeObject(heroi);//removendo do cenário
        heroi = new SuperHeroi();//promovendo o objeto heroi ao status de SuperHeroi
        this.addObject(heroi,x, y);//reintroduzindo no cenário
        heroi.setPontos(p);//repassando pontuação salva
        heroi.setRotation(r);//repassando rotação salva
    }
}
```

Fonte: do autor.

A partir da análise, avaliação e tabulação das pontuações obtidas pelos estudantes, constatou-se diferenças significativas nos resultados em ambas as turmas. No que se refere a dispersão dos dados (Figuras 17 e 18), percebe-se que na Turma Piloto as pontuações dos estudantes ficaram mais dispersas na segunda amostra. Já na Turma 2018, as pontuações ficaram menos dispersas na segunda amostra. Em geral, essa constatação indica que na Turma Piloto, alguns estudantes avançaram bem mais que a maioria, caso dos estudantes A9, A16 e A26, aumentando o valor do desvio padrão médio. Em relação à Turma 2018, a redução do desvio padrão indica que a turma se tornou mais homogênea no que se refere à compreensão dos conceitos de POO. É fundamental ressaltar que, em ambos os casos, o desvio padrão tem a finalidade de medir a dispersão da turma em relação à apropriação dos conceitos de POO e, em ambos os casos, o resultado foi positivo, visto que a média global das turmas aumentou.

Verifica-se também a partir da análise das pontuações dos estudantes nas turmas Piloto e 2018, avanços individuais significativos, pois, na Turma Piloto 89% dos estudantes aumentou as suas pontuações na segunda amostra e, na Turma 2018 foram 80% a obter aumento. Ainda, no que se refere aos estudantes que tiveram suas médias reduzidas (11% na Turma Piloto e 20% na Turma 2018), é possível se considerar que os decréscimos foram significativamente baixos, especialmente na Turma 2018, em que o decréscimo médio foi de 7,5%.

No tocante às respostas do questionário de opinião (APÊNDICE G) aplicado com ambas as turmas, verifica-se que o Greenfoot foi eficiente no sentido de oferecer condições de aprendizagem dos conceitos de POO, pois, segundo a percepção dos estudantes, na Turma Piloto, 83,3% respondeu ter visualizado “muitos” dos conceitos de POO e 4,2% afirmam ter observado “todos” os conceitos. Na Turma 2018, o Greenfoot foi melhor avaliado, pois 60% dos estudantes afirmam ter observado “todos” e 40% “muitos” dos conceitos de POO.

Ainda, no que se refere a compreensão dos conceitos de POO, na Turma Piloto, 62,5% dos estudantes afirma que o Greenfoot facilitou “razoavelmente”, 25% indicou que facilitou “plenamente” e 12,5% considerou o uso do Greenfoot “indiferente” para aprender POO. Já na turma 2018, as respostas foram mais positivas nesse aspecto, pois 45% apontou que o uso do Greenfoot facilitou “razoavelmente” e 55% afirmaram que facilitou “plenamente” a compreensão dos conceitos de POO.

A boa aceitação dos estudantes à proposta pode ser verificada quando solicitado a eles para definir uma nota de 1 a 4 às atividades realizadas com o uso do Greenfoot, nesse caso, a grande maioria atribuiu notas 3 (54,2% e 40%) ou 4 (45,8% e 60%). Notas 1 ou 2 não foram assinaladas por nenhum dos estudantes em ambas as turmas.

Em relação às opiniões manifestadas pelos estudantes, percebeu-se que na turma Piloto houve quatro (22%) comentários sugestivos, quatro (22%) comentários negativos e dez comentários positivos (56%). Já na turma 2018, houve a ocorrência de oito (73%) comentários positivos, uma (9%) sugestão e duas (18%) críticas à ferramenta. Assim, na turma 2018 não foram emitidos comentários negativos, o que demonstra que ao adequar as atividades às sugestões e comentários negativos da Turma Piloto a dinâmica foi mais eficiente e melhor na turma 2018.

Nesse sentido, constata-se que a decisão de avaliar a intervenção piloto e reestruturar as dinâmicas foi fundamental para o sucesso da proposta, visto que as colocações dos estudantes da Turma Piloto serviram de base à tomada de decisão do docente para as próximas atividades. Nesse caso, o planejamento de intervenções menos cansativas e com maior carga horária dedicada para a sua realização, possibilitou ao docente dispensar atenção adequada a todos os alunos. Trata-se de um movimento fundamental do fazer pedagógico, pois às constatações oriundas da práxis são a base para o aprimoramento da prática docente, que se dá pela aceitação do contraditório e da autonomia do educando, visto que, cada estudante tem um ritmo de aprendizagem e compete ao docente perceber tais particularidades e adequar suas dinâmicas (FREIRE, 1996).

Tais ações possivelmente impactaram positivamente na segunda intervenção, visto que os dados coletados no formulário indicam aumento significativo nas avaliações positivas dos estudantes em relação às dinâmicas realizadas. Outro ponto que certamente contribuiu de maneira positiva, está relacionado a experiência do docente frente à proposta, pois estava aplicando pela segunda vez e, nesse caso, detém a visão do todo e reconhece mais claramente as possibilidades e potencialidades do Greenfoot.

É importante salientar que o Ciclo de Aprendizagem de Kolb (KOLB, 1984) esteve presente nas atividades desenvolvidas em todas as unidades de estudo. O elemento inicial do ciclo é evidenciado no início da atividade, pois “*experiência concreta*” está relacionada com as sensações que os estudantes têm a partir do

contato com a proposta e ao assistir o vídeo explicativo do jogo, pois proporciona que tenham experiência relacionada a fazer uma tarefa, mesmo que em termos de observação e sentimento – não necessariamente uma abordagem teórica e sistemática (Marieto et al, 2014).

A partir das sensações que os estudantes tiveram, seja pelo contato com a proposta ou pela visualização do vídeo, ocorre, em suas consciências, o que Kolb (1984) chamou de "*observação reflexiva*", momento em que os estudantes param para refletir acerca do que visualizaram e preparam-se para realizar algum tipo de ação prática - esse estágio não inclui necessariamente alguma ação (Marieto et al, 2014).

A "*conceitualização abstrata*" acontece depois de passados os dois estágios anteriores, e é caracterizada pelo momento de agir frente a proposta, buscar solucionar o problema a que estão diante – programar o jogo. Nesse momento, os estudantes colocam em prática seus conhecimentos, formulam hipóteses, enfim, dedicam-se em buscar uma solução à luz de sua carga cognitiva. O passo seguinte é testar o efeito da sua 'proposta' de solução para o problema em tela – esse momento foi chamado por Kolb (1984) de "*experimentação ativa*", pois o estudante coloca à prova os seus conhecimentos e a sua hipótese de solução. Notadamente, tais atitudes se fecham em ciclo constante, pois ao testar as suas teorias, hipóteses e práticas, os estudantes ficam sujeitos a novas sensações oportunizadas pela observação e constatação do que aconteceu, voltando, dessa maneira, ao primeiro estágio do ciclo ("*experiência concreta"*).

Esse movimento constante, em um ciclo contínuo foi verificado durante toda a realização da pesquisa, pois, até que os estudantes chegassem ao final das atividades, necessitaram, por muitas vezes recomeçar este "*loop"* no processo de aprendizagem. De certa forma, no que se refere aos passos de cada intervenção (Figura 8), constata-se que o pesquisador realiza o mesmo processo, pois ao colocar-se diante da problemática, tem-se a "*experiência concreta"* e ao pensar em como solucionar tal problema, realiza uma "*observação reflexiva"*. Ao formular uma proposta de solução, constrói hipóteses e coloca em prática seus conhecimentos – elementos característicos da "*conceitualização abstrata"* e, por fim, testa a sua proposta (*experimentação ativa*) que produzirá novas sensações e o farão retornar ao início do ciclo (*experiência concreta*).

Esse movimento de "tentativa e erro/acerto" é muito característico da programação de computadores, mas está também, muito presente nos jogos, pois

nem sempre os jogadores conseguem completar o objetivo na primeira tentativa e, a atitude de recomeçar, tentar hipóteses diferenciadas de solução culmina, na maioria das vezes, em vitória – porém, ao perder ou vencer o desafio, ocorre aprendizagem.

6 CONSIDERAÇÕES FINAIS

O ensino e aprendizagem de programação não é uma tarefa trivial no cotidiano dos cursos de computação. Muitos são os fatores que influenciam nas dificuldades dos estudantes em assimilar os conceitos relacionados ao desenvolvimento de software. Alguns desses fatores foram mencionados no decorrer dessa dissertação, com destaque à pesquisa de Souza, Batista e Barbosa (2016), a qual aponta para dificuldade em abstrair problemas cotidianos e, a partir deles, planejar e construir soluções computacionais eficientes.

Além disso, a área da programação de computadores dispõe de diferentes técnicas de desenvolvimento e, dentre elas a Programação Orientada a Objetos – foco desse trabalho. Tal técnica de desenvolvimento, constantemente, é caracterizada pela dificuldade de assimilação por parte dos estudantes, pois além de carregar todos os elementos da programação estruturada (procedural), agrega outros elementos que, embora, tragam grandes benefícios ao processo de desenvolvimento de software, nem sempre são fáceis de compreender.

De outro lado, tem-se uma geração de jovens, futuros desenvolvedores, imersos no contexto tecnológico contemporâneo, chamado de Sociedade da Informação por Coll e Monereo (2010), com comportamentos peculiares descritos por Veen e Vrakking (2009) e Prensky (2001; 2012), naturalmente amantes de jogos (Papert, 2008) e consumidores de tecnologia - pouco interessados em compreendê-la em profundidade.

Diante desse contexto, a presente pesquisa teve o objetivo de compreender como a utilização do desenvolvimento de jogos pode contribuir na apropriação dos conceitos de programação orientada a objetos. Nesse sentido, para alcançar tal propósito, realizou-se um levantamento bibliográfico, conduzido, primeiramente pelo referencial teórico, com a finalidade de destacar os princípios norteadores desse trabalho. A partir do referencial teórico, foram detalhados aspectos relacionados ao processo histórico de constituição das principais linguagens e técnicas de programação, indicando que a POO é um dos paradigmas mais utilizados na atualidade. Outro aspecto levantado nessa fase da pesquisa refere-se a importância de estudar conceitos de computação, pois o cenário tecnológico converge para, em

um futuro próximo, a ampliação dos postos de trabalho na área de tecnologia frente a tendência de utilização pervasiva de equipamentos computacionais.

No que se refere à área da educação, o referencial teórico destaca a influência das TIC no cotidiano escolar, auxiliando nos processos administrativos e educativos, aproximando o fazer pedagógico do contexto social e, sobretudo tecnológico. Ainda, sobre processos de ensino e aprendizagem, o uso de jogos ganha destaque frente a sua capacidade de envolver um público que se encontra cada vez mais disperso quando o assunto é educação.

Na sequência da pesquisa, uma revisão sistemática da literatura (RSL) foi realizada com o objetivo de levantar o “estado da arte”, em nível nacional, no que se refere a ferramentas computacionais e referencial metodológico de auxílio ao ensino de programação de computadores, com ênfase em POO. Nesse caso, “aprendizagem por projetos” e o uso de “jogos” recebeu maior destaque nos trabalhos pesquisados, ainda, o software Alice e Robocode foram os únicos explorados para o ensino de POO. Tendo em vista os poucos resultados obtidos no que se refere a ferramentas para o ensino de POO, a revisão foi expandida para testes práticos em outras opções. Nesse caso, o Greenfoot, embora tenha sido pouco explorado nos trabalhos levantados na RSL, apresentou boas possibilidades de utilização.

O referencial metodológico compreendeu o desenvolvimento de uma pesquisa-ação para alcançar o objetivo da pesquisa. Tal procedimento foi organizado pela utilização conjunta de Unidades de Estudo, Greenfoot, Mapas Conceituais e Ciclo de Aprendizagem de Kolb em duas intervenções pedagógicas em turmas concluintes do Curso Técnico em Informática Integrado ao Ensino Médio do IFFAR – *Campus* Santo Augusto.

Foram produzidas seis unidades de estudos para organizar cada intervenção, sendo que, este recurso instrucional se mostrou eficiente em traçar um itinerário pedagógico para o ensino de POO. Do mesmo modo, a IDE Greenfoot ofereceu suporte adequado à proposta, pois disponibilizou os elementos necessários para proporcionar condições de aprendizagem de POO. A possibilidade de interação em nível de código-fonte, *feedback* instantâneo, uso da linguagem java (OO) sob a perspectiva do desenvolvimento de jogos, tornaram o Greenfoot a ferramenta ideal para esta pesquisa.

Os mapas conceituais atuaram como instrumento de coleta de informações acerca da evolução cognitiva dos estudantes em relação à POO, pois foram

construídos mapas conceituais antes e depois da realização das atividades descritas nas unidades de estudos. O Ciclo de Aprendizagem de Kolb esteve em evidência durante todo o processo, pois, as dinâmicas utilizadas nas atividades realizadas convergem para o fluxo contínuo de *experiência concreta, observação reflexiva, conceitualização abstrata* e *experimentação ativa.*

Os dados obtidos pela avaliação dos mapas conceituais antes e depois de cada intervenção mostrou-se resultados positivos em ambas as turmas pesquisadas, pois se pôde verificar evolução das médias individuais, comprovada matematicamente pela realização de teste estatístico (teste t). Do mesmo modo, a maioria dos estudantes avaliou positivamente o processo envolvendo o uso do Greenfoot para a aprendizagem de POO.

É fundamental, também, nesse momento explicitar o crescimento do profissional docente no que se refere às práticas de ensino de POO, pois a partir da realização da pesquisa foi possível apresentar aos estudantes uma proposta mais dinâmica que, embora demande trabalhoso planejamento, produz bons resultados. Perceber o envolvimento dos estudantes na realização das tarefas significa a materialização do dever cumprido.

Por outro lado, ao constatar que nem todos os estudantes compreendem que essa é a melhor forma de se aprender, é preciso refletir e aceitar que no espaço educativo, convivemos com uma pluralidade de opiniões e perfis. Nesse caso, o profissionalismo deve prevalecer e, agir no sentido de buscar alternativas que contemplem a totalidade do público discente.

Diante de todos os elementos descritos até aqui, é possível considerar positiva a utilização do conjunto de materiais: Unidades de Estudo, Greenfoot, Mapas Conceituais e Ciclo de Aprendizagem de Kolb para o ensino de POO, sobretudo, a partir da dinâmica do desenvolvimento de jogos. Tal constatação se deve pela boa aceitação dos estudantes à proposta, baseada nas avaliações positivas expressadas no questionário de opinião e, sobretudo, pelos avanços na aprendizagem de POO observados na tabulação dos mapas conceituais de ambas as turmas participantes, formalizada matematicamente através da realização de teste estatístico (seção 5.3).

Como trabalhos futuros, uma proposta é utilizar os mapas conceituais dos estudantes para identificar os conceitos de POO que estiveram em maior e menor evidência, pois, dessa maneira será possível aprimorar as unidades de estudo no sentido de dar maior ênfase a conceitos menos assimilados.

Outra atividade com potencial para trabalhos futuros está relacionada a traçar o perfil da turma, a partir dos Estilos de Aprendizagem de Kolb, a fim de verificar quais estilos de aprendizes (KOLB,1984) apresentam maiores facilidades e dificuldades em assimilar conceitos de POO e, a partir disso, buscar alternativas eficientes para a prática educativa.

REFERÊNCIAS

AMARAL, L., BRAGA E SILVA, G., & PANTALEÃO, E. (2015). Plataforma robocode como ferramenta lúdica de ensino de programação de computadores - pesquisa e extensão universitária em escolas públicas de Minas Gerais. **Anais do XXVI Simpósio Brasileiro de Informática na Educação**, 200–208.

ANSELMO, Fernando. **Aplicando a Lógica Orientada a Objetos em Java**: da lógica à certificação. 3.ed. Florianópolis: Visual Books, 2013.

ANTONIO, E. A., FERRO, M.,: **Uma Abordagem para o Ensino de Orientação a Objetos apoiada pelo Ambiente Virtual MOODLE, *software* Alice, PBL e Mapas**. In: MoodleMoot Brasil 2010, São Paulo — SP (2010).

BARBIER, René. **A pesquisa-Ação**. Brasília: Liber Livros Editora, 2007.

BARBOSA, Luciana Leal da Silva; SANTOS, Gabriel; CRUZ, Marion da Silva. **Projeto Investindo em Novos Talentos: Introdução à programação para crianças com o Alice 3D**. Alice Brasil, 2016, São Paulo, **Anais...** São Paulo: Páginas & Letras, 2016.

BECKER, Fernando. **Educação e Construção do Conhecimento**. 2.ed. Porto Alegre: Penso, 2012.

BRACKMANN, C. P**. Desenvolvimento do pensamento computacional através de atividades desplugadas na educação básica**. 2017. 226f. Tese (Doutorado em Informática na Educação) ─ Universidade Federal do Rio Grande do Sul, Porto Alegre, 2017.

BRANDÃO, Jammilly Mikaela Fagundes. **Princípios andragógicos e fatores mediadores da aprendizagem na educação a distância em administração pública.** 2014. 193f. Dissertação (Mestrado em Administração) - Universidade Federal da Paraíba. João Pessoa.

BRASIL. Ministério da Educação. **Plano de Desenvolvimento Institucional**. Instituto Federal de Educação, Ciência e Tecnologia Farroupilha, 2014. Disponível em: <http://www.iffarroupilha.edu.br>. Acesso em: dez. 2017.

CARDOSO, Caíque. **Orientação a objetos na prática:** aprendendo orientação a objetos com Java. Rio de Janeiro: Ciência Moderna, 2006

CARVALHO, V. A. de. TEIXEIRA, G. F. **Programação orientada a objetos**: Curso técnico de informática. Colatina: IFES, 2012.

CASTELLS, Manuel. **A sociedade em rede**. 17. ed. São Paulo: Paz e Terra, 2016.

CEMED, Centro de Estudos do Medicamento**. Mapas conceituais - O que são - Mapa conceitual de Mapa conceitual**. ebah.com.br. 02/08/2010. Disponível em: http://www.ebah.com.br/content/ABAAAfUq4AK/mapas-conceituais-que-sao-mapa-conceitual-mapa-conceitual>. Acesso em: 16 set 2017.

CLARETO, Sônia Maria. **MATEMÁTICA COMO ACONTECIMENTO NA SALA DE AULA**. 36ª Reunião Nacional da ANPEd. 2013, Goiânia. Disponível em: <http://36reuniao.anped.org.br/pdfs_trabalhos_aprovados/gt19_trabalhos_pdfs/gt19_3248_texto.pdf>. Acesso em 11 jun 2018.

COLL, César; MONEREO, Carles. **Psicologia da educação virtua**l: aprender e ensinar com as tecnologias da informação e da comunicação. Porto Alegre: Artmed, 2010.

DA ROSA, Rosane Teresinha Nascimento; LORETO, Élgion Lúcio Silva. Análise, através de mapas conceituais, da compreensão de alunos do ensino médio sobre a relação DNA-RNA-PROTEÍNAS após o acesso ao GenBank. **Investigações em Ensino de Ciências**, v. 18, n. 2, p. 385-405, 2013.

DEITEL, P.; DEITEL, H. **Java**: como programar. São Paulo: Pearson Education do Brasil, 2017.

FARDO, Marcelo Luis. A Gamificação Aplicada em Ambientes de Aprendizagem. **Revista Renote: novas tecnologias na educação**, V.11, Nº 1, julho, 2013.

FERREIRA, Giselle Martins dos Santos; CASTIGLIONE, Rafael Guilherme Mourão. **TIC na educação: ambientes pessoais de aprendizagem nas perspectivas e práticas de jovens**. Educ. Pesqui., São Paulo, 2017 . Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1517-97022017005009101&lng=en&nrm=iso>. Acesso em: 30 Nov. 2017.

FILATRO, Andrea; CAIRO, Sabrina. **Produção de conteúdos educacionais**. São Paulo: Saraiva, 2015.

FREIRE, Paulo. **Pedagogia da autonomia**: saberes necessários à prática educativa. São Paulo: Paz e Terra, 1996.

GIL, A. C. **Como elaborar projetos de pesquisa**. 5.ed. São Paulo: Atlas, 2010.

GREENROON. **The Greenfoot Educators Community.** 2017. Disponível em: < https://greenroom.greenfoot.org/door>. Acesso em: 14 out 2017.

HAMMER, Øyvind. **PAST**: Reference Manual (2018). Disponível em: < http://folk.uio.no/ohammer/past>. Acesso em: 23 jul 2018.

HAMMER, Øyvind. **Past 3.x:** the Past of the Future (2018b). Disponível em: < http://folk.uio.no/ohammer/past>. Acesso em: 23 jul 2018.

HUIZINGA, Johan. **Homo ludens**: o jogo como elemento da cultura. São Paulo: Perspectiva, 2007.

IEEE. **About IEEE**. 2017. Disponível em: <http://www.ieee.org/about/about_index.html>. Acessado em: 26 set 2017b.

IEEE. **Interactive:** The Top Programming Languages 2017. Disponível em: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017. Acesso em: 26 set 2017a.

JESUS, A. M.; GONÇALVES, D. A. S.; FERREIRA, L. A. C. **Aplicação de Desenvolvimento de Jogos Digitais como um Meio de Motivação em Diferentes Níveis de Ensino de Computação**. In: XX Workshop de Informática na escola – WIE, 2014, Dourados, **Anais...** Dourados: SBC, 2014, p.56-65.

JOYANES AGUILAR, Luis. **Fundamentos de programação**: algoritmos, estruturas de dados e objetos. São Paulo: McGraw-Hill, 2008.

KEOGH, James Edward; GIANNINI, Mario. **OOP desmitificado:** programação orientada a objetos. Rio de Janeiro: Alta Books, 2005.

KITCHENHAM, B. **Guidelines for performing Systematic Literature Reviews in Software Engineering**, version 2.3. Technical Report EBSE. Software Engineering Group. School of Computer Science and Mathematics Keele University. 2007.

KOLB, D**. Experiential Learning:** experience as the source of learning and development. New Jersey: Prentice Hall, 1984.

KÖLLING, M. The greenfoot programming environment. **ACM Transactions on Computing Education (TOCE)**, v.10, n.4. 2010. Disponível em: https://dl.acm.org/citation.cfm?id=1868361. Acesso em: 13 out 2017.

LARSON, Ron. **Estatística aplicada**. 2. ed. São Paulo: Pearson Prentice Hall, 2007.

LARSON, Ron; FARBER, Betsy. **Estatística Aplicada**. 4.ed. São Paulo: Pearson Prentice Hall, 2010.

LÉVY, Pierre. **O Que é o Virtual:** Pierre Lévy. São Paulo: Editora 34, 1996.

MARIETTO, M. das G. B. et al. **Teoria da Aprendizagem Experiencial de Kolb e o Ciclo de Belhot Guiando o Uso de Simulações Computacionais no Processo Ensino Aprendizagem**. In: XX Workshop de Informática na escola – WIE, 2014, Dourados, Anais... Dourados: SBC, 2014, p.527-531.

MARTINS, Renata Lacerda Caldas; VERDEAUX, Maria de Fátima da Silva; SOUSA, Célia Maria Soares Gomes de**. A utilização de diagramas conceituais no ensino de física em nível médio**: um estudo em conteúdos de ondulatória, acústica e óptica. Rev. Bras. Ensino Fís., São Paulo, v.31, n.3, p. 3401.1-3401.12. 2009. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1806-11172009000300005&lng=en&nrm=iso>. Acessado em: 20 Set. 2017.

MATIAS-PEREIRA, J. **Manual de Metodologia da Pesquisa Científica**. 3. Ed. São Paulo: Atlas, 2012.

MATTAR, João. **Games em educação**: como os nativos digitais aprendem. São Paulo: Pearson, 2009.

MIRANDA, N. X. D. **Aprender a cómo yo y los demás aprendemos**. 2011. Disponível em: <https://buildership.wordpress.com/2011/09/26/nuestras-preferencias-al-aprender/> . Acesso em: 04 out. 2017.

MONTEIRO, Mario A. **Introdução à organização de computadores**. 5. ed. Rio de Janeiro: LTC, 2007

MOORE, David S. **A estatística básica e sua prática**. 3. ed. Rio de Janeiro: LTC, 2005.

MORAES, Maria Cândida. **O Paradigma Educacional Emergente**. 14. ed. São Paulo: Papirus, 1997

MOREIRA, Marco A. **Teorias de aprendizagem**. 2. ed. ampl. Rio de Janeiro: EPU, 2011.

MOREIRA, Marco Antonio; MASINI, Elcie F. S. **Aprendizagem Significativa**: a teoria de David Ausubel. 2ª ED. 2006. 1ª Reimpressão. São Paulo: Centauro, 2009.

NOVAK, J. D.; GOWIN, D. B. **Aprender a Aprender**. Lisboa: Plátano Edições Técnicas. 1996.

OMG. **About OMG**. 2017. Disponível em: < http://www.omg.org/about/index.htm>. Acessado em: 06 nov 2017.

PAPERT, Seymour M. **A Máquina das Crianças: Repensando a escola na era da informática** (edição revisada). Porto Alegre: Artmed, 2007.

_____. **The Children's Machine**: rethinking school in the age of the computer. Basic Books. New York, 1992.

_____. **Logo**: Computadores e Educação. São Paulo: Brasiliense, 1985.

PEÑA, Antonio O. et al. **Mapas Conceituais**: uma técnica para aprender. São Paulo: Loyola, 2005.

PIMENTEL, Alessandra**. A teoria da aprendizagem experiencial como alicerce de estudos sobre desenvolvimento profissional**. Estud. psicol. (Natal), Natal, v. 12, n. 2, p. 159-168. 2007. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1413-294X2007000200008&lng=en&nrm=iso>. Acesso em: 02 Nov. 2017.

PONCE, Julio V. **Distribución normal** (2018). Disponível em: <http://calidadcliente.blogspot.com/2017/10/distribucion-normal.html>. Acesso em: 24 jul 2018.

PORTAL ACTION. **Action Stat**. Disponível em: <http://www.portalaction.com.br/sobre-o-action>. Acesso em: 22 jul 2018.

PRENSKY, Marc. Digital natives, digital immigrants. **On the Horizon**, Minneapolis, v. 9, n. 5, p. 1-6, 2001. Disponível em: <http://www.emeraldinsight.com/doi/pdfplus/10.1108/10748120110424816> . Acesso em: 13 nov. 2017.

_____. Aprendizagem Baseada Em Jogos Digitais. São Paulo: SENAC, 2012.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Elsevier, 2003.

SCAICO, Pasqualine D. et al. Ensino de Programação no Ensino Médio**:** Uma Abordagem Orientada ao Design com a linguagem Scratch. **Revista Brasileira de Informática na Educação**, v.21, n.2, p. 92-103, 2013.

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

SHIMOHARA, C.; SOBREIRA, E. S. R. **Criando Jogos Digitais para a aprendizagem de matemática no ensino fundamental I**. In: XXI Workshop de Informática na escola – WIE, 2015, Maceió, **Anais...** Maceió: SBC, 2015, p.72-81.

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

SOUZA, Alex S. A. et al. **Uso do Scratch e do Alice para o ensino de lógica na rede pública de ensino do Litoral Norte do Estado de São Paulo**. Alice Brasil, 2016, São Paulo, **Anais...** São Paulo: Páginas & Letras, 2016.

SOUZA, Draylson Micael; BATISTA, Marisa Helena da Silva; BARBOSA, Ellen Francine. Problemas e Dificuldades no Ensino e na Aprendizagem de Programação: Um Mapeamento Sistemático. **Revista Brasileira de Informática na Educação**, v.24, n.1, p. 39-52, 2016.

SOUZA, Marcella S. Café et al. **Lord of Code:** uma ferramenta de apoio ao ensino de programação. In: XXVII Simpósio Brasileiro de Informática na Educação – SBIE, 2016, Uberlândia, **Anais...** Uberlândia: SBC, 2016, p.1316-1320.

TAJRA, Sanmya Feitosa. **Informática na educação**: Novas ferramentas pedagógicas para o professor na atualidade. 8. ed. rev. e ampl. São Paulo: Érica, 2009.

TEIXEIRA, Adriano C. et al. **Programação de computadores para alunos do ensino fundamental:** A Escola de Hackers. In: XXI Workshop de Informática na escola – WIE, 2015, Maceió, **Anais...** Maceió: SBC, 2015, p.112-121.

THIOLLENT, M. **Metodologia da pesquisa-ação**. 18. ed. São Paulo: Cortez, 2011.

TORMAN, Vanessa Bielefeldt Leotti; COSTER, Rodrigo; RIBOLDI, João. **Normalidade de variáveis**: métodos de verificação e comparação de alguns testes não-paramétricos por simulação. Clinical & Biomedical Research, [S.l.], v. 32, n. 2, jul 2012. ISSN 2357-9730. Disponível em: <http://seer.ufrgs.br/index.php/hcpa/article/view/29874>. Acessado em: 23 jul 2018.

TUCKER, Allen B.; NOONAN, Robert E. **Linguagens de programação**: princípios e paradigmas. 2.ed. Porto Alegre : AMGH, 2010.

VALENTE, J. A. (org) **Computadores e Conhecimento: repensando a educação**. Campinas, S.P: Gráfica Central da UNICAMP, 1993.

VALENTE, José Armando. Integração do pensamento computacional no currículo da educação básica: diferentes estratégias usadas e questões de formação de professores e avaliação do aluno. **Revista e-Curriculum**, São Paulo, v.14, n.03, p. 864 –897 jul./set.2016.

VEEN, Wim; VRAKKING, Ben. **Homo zappiens**: educando na era digital. Porto Alegre: Artmed, 2009.

VIGOTSKI, L.S. **Pensamento e Linguagem.** Lisboa: Relógio D'Água, 2007.

WANGENHEIM, Christiane Gresse von; NUNES, Vinícius Rodrigues; SANTOS, Giovane Daniel dos. Ensino de Computação com SCRATCH no Ensino Fundamental: Um Estudo de Caso. **Revista Brasileira de Informática na Educação**, v.22, n.3, p. 115-125, 2014.

WANGENHEIM, Christiane Gresse von; WANGENHEIM, Aldo von. **Ensinando computação com jogos**. Florianópolis: Bookess Editora, 2012.

WING, J. M. (2008). Computational thinking and thinking about computing. **Phil. Trans. R. Soc. A**, 366(1881):3717–3725.

APÊNDICE A – UNIDADE DE ESTUDO 1.1

Aprendendo Orientação a Objetos com Greenfoot

Unidade de Estudo 1.1

{Abstração, Classe, Objeto }

1 Objetivos

- Compreender o que são objetos computacionais, classes, atributos, métodos e instanciação;
- Entender a relação entre classes e objetos;
- Compreender a importância da abstração para a implementação de software.

2 Papéis

Estudantes organizam-se em grupos e professor atua como mediador da proposta.

3 Duração

4h aula;

4 Conteúdos

Anexo 1 – Apresentação POO:
https://drive.google.com/open?id=1Bixy_dKOmVW2DrJdMnRrFZ5cri2ICF96

Anexo 2 – Apostila POO:
https://drive.google.com/open?id=1TOytzDQLnsovZTDdwu7kZlfRf7AeIT5c

Anexo 3 - Tutorial Greenfoot:
https://www.greenfoot.org/files/translations/Brazilian/Tutorial%20do%20Greenfoot.htm

Anexo 4 - Banco de imagens:
https://drive.google.com/drive/folders/1zZTYKyKIcAUNMwGyCbDC-fx6_FNnnqgI?usp=sharing

5 Atividades

Leia atentamente ao texto abaixo:

O automóvel como um objeto

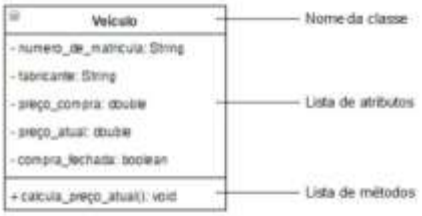
"Para ajudar a entender objetos e seus conteúdos, vamos começar com uma analogia simples. Suponha que você queira guiar um carro e fazê-lo andar mais rápido pisando no pedal acelerador. O que deve acontecer antes que você possa fazer isso? Bem, antes de poder dirigir um carro, alguém tem de projetá-lo. Um carro tipicamente começa como desenhos de engenharia, semelhantes a plantas que descrevem o projeto de uma casa. Esses desenhos incluem o projeto do pedal do acelerador. O pedal oculta do motorista os complexos mecanismos que realmente fazem o carro ir mais rápido, assim como o pedal de freio "oculta" os mecanismos que diminuem a velocidade do carro e a direção "oculta" os mecanismos que mudam a direção dele. Isso permite que pessoas com pouco ou nenhum conhecimento sobre como motores, freios e mecanismos de direção

funcionam consigam dirigir um carro facilmente. Assim como você não pode cozinhar refeições na planta de uma cozinha, não pode dirigir os desenhos de engenharia de um carro. Antes de poder guiar um carro, ele deve ser construído a partir dos desenhos de engenharia que o descrevem. Um carro pronto tem um pedal de acelerador real para fazê-lo andar mais rápido, mas mesmo isso não é suficiente — o carro não acelerará por conta própria (tomara!), então o motorista deve pressionar o pedal do acelerador" (DEITEL & DEITEL, 2016, p.8).

*"A **abstração** é uma característica fundamental da programação e do projeto orientados a objetos, pois é por meio dela que o projetista a ideia geral da proposta e dos objetivos do produto a ser desenvolvido. Essa característica permite que grandes sistemas sejam especificados em um nível global, antes de ocorrer a implementação dos métodos individuais"* (TUCKER e NOONAM, 2010)."

"Previamente à implementação de uma classe, é realizada a sua modelagem conforme regras estabelecidas pelos padrões de desenvolvimento de software, que atualmente é realizada pela UML, a qual tem suas especificações padronizadas pela OMG[1]. A Figura 1 ilustra a representação UML de uma classe hipotética 'Veículo'.

Figura 1 - Representação de uma classe



Assim, como se pode verificar na Figura 4, cada objeto criado a partir dessa classe, terá os mesmos atributos e métodos nela descritos. O que será diferente de um objeto em relação a outro será o seu identificador e o seu "estado" que é formado pelos valores armazenados em seus atributos."

Observe a Figura 2 e Vídeo 1 em anexo e responda às questões do item 7:

[1] O Object Management Group® (OMG®) é uma associação aberta internacional, sem fins lucrativos, que tem o objetivo de desenvolver padrões internacionais de tecnologia para usuários do mundo inteiro. Essa associação foi fundada em 1989 e, desde então, fornece padrões OMG direcionados à fornecedores, usuários finais, instituições acadêmicas e agências governamentais (OMG, 2017).

Aprendendo Orientação a Objetos com Greenfoot

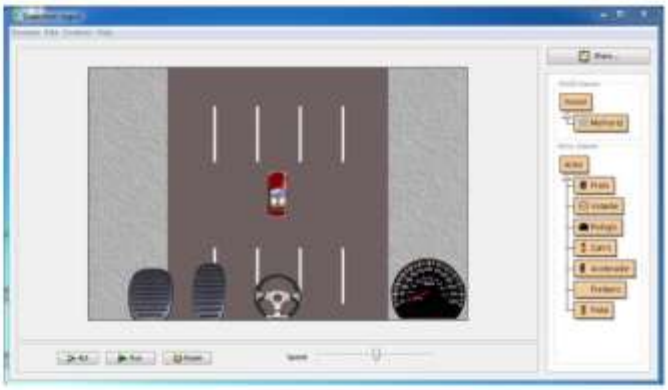
Figura 2 - Imagem Simulador

Video 1 - Video Simulador

https://youtu.be/dIkNHQPIlv0

Atividade: Utilizando o Greenfoot, procure construir a interface visualizada na Figura 3.

Figura 3 - Prospecto do Simulador



6 Ferramentas

- Software Greenfoot;
- Software Draw.io Diagrams

7 Avaliação

Questões

1) Analisando a figura e vídeo, quais objetos podemos identificar?
2) Em relação à POO, o que são objetos?
3) Os objetos precisam ser modelados (descritos) – *"antes de poder dirigir um carro, alguém tem de projetá-lo"(DEITEL & DEITEL, 2016, p.8)* - para que sejam determinadas as suas características e funcionalidades dentro do sistema. Assim, muitos objetos pertencem a um mesmo "grupo". Em POO, esse grupo é denominado?

4) Ao estudarmos POO, aprendemos que a primeira fase do desenvolvimento de um software sob a abordagem de POO, inicialmente construímos a modelagem das classes a serem implementadas. Que nome é atribuído a esse processo? O que é importante observar nesse processo?
5) Ao analisar o vídeo, quais deles exercem algum tipo de ação? Descreva de maneira literal essas ações.
6) Utilizando o software Draw.io construa os diagramas UML das classes (ver) observadas na Figura 1 e Figura 3 e no Vídeo 1 (anexo)?

Referências

CARDOSO, Caíque. **Orientação a objetos na prática:** aprendendo orientação a objetos com Java. Rio de Janeiro: Ciência Moderna, 2006

CARVALHO, V. A. de. TEIXEIRA, G. F. **Programação orientada a objetos**: Curso técnico de informática. Colatina: IFES, 2012.

DEITEL, P.; DEITEL, H. **Java**: como programar. São Paulo: Pearson Education do Brasil, 2017.

JOYANES AGUILAR, Luis. **Fundamentos de programação**: algoritmos, estruturas de dados e objetos. São Paulo: McGraw-Hill, 2008.

KEOGH, James Edward; GIANNINI, Mario. **OOP desmitificado:** programação orientada a objetos. Rio de Janeiro: Alta Books, 2005.

OMG. **About OMG**. 2017. Disponível em: < http://www.omg.org/about/index.htm>. Acessado em: 06 nov 2017.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Elsevier, 2003.

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

TUCKER, Allen B.; NOONAN, Robert E. **Linguagens de programação**: princípios e paradigmas. 2.ed. Porto Alegre : AMGH, 2010.

APÊNDICE B – UNIDADE DE ESTUDO 1.2

Aprendendo Orientação a Objetos com Greenfoot

Unidade de Estudo 1.2

{Mensagem, Atributos, Métodos, construtor e Instanciação}

1 Objetivos

Compreender o que é Método, atributo, instanciação e envio/recebimento de mensagem, construtor;

2 Papéis

Estudantes organizam-se em grupos e professor atua como mediador da proposta.

3 Duração

3h aula;

4 Conteúdos

Anexo 1 – Apresentação POO:
https://drive.google.com/open?id=1Bixy_dKOmVW2DrJdMnRrFZ5cri2ICF96

Anexo 2 – Apostila POO:
https://drive.google.com/open?id=1TOytzDQLnsovZTDdwu7kZlfRf7AeIT5c

Anexo 3 - Tutorial Greenfoot:
https://www.greenfoot.org/files/translations/Brazilian/Tutorial%20do%20Greenfoot.htm

Anexo 4 - Banco de imagens:
https://drive.google.com/drive/folders/1zZTYKyKIcAUNMwGyCbDC-fx6_FNnnqgI?usp=sharing

5 Atividades

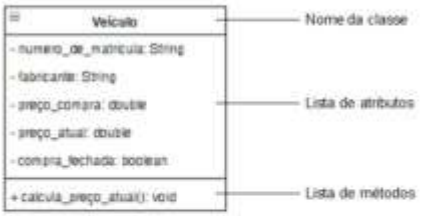
Leia atentamente ao texto abaixo:

A **abstração** é uma característica fundamental da programação e do projeto orientados a objetos, pois é por meio dela que o projetista a ideia geral da proposta e dos objetivos do produto a ser desenvolvido. Essa característica permite que grandes sistemas sejam especificados em um nível global, antes de ocorrer a implementação dos métodos individuais" (TUCKER e NOONAM, 2010).

Previamente à implementação de uma classe, é realizada a sua modelagem conforme regras estabelecidas pelos padrões de desenvolvimento de software, que atualmente é realizada pela UML, a qual tem suas especificações padronizadas pela OMG[1]. A Figura 1 ilustra a representação UML de uma classe hipotética 'Veículo'.

[1] O Object Management Group® (OMG®) é uma associação aberta internacional, sem fins lucrativos, que tem o objetivo de desenvolver padrões internacionais de tecnologia para usuários do mundo

Figura 1 - Representação de uma classe



Assim, como se pode verificar na Figura 1, cada objeto criado a partir dessa classe, terá os mesmos atributos e métodos nela descritos. O que será diferente de um objeto em relação a outro será o seu identificador e o seu "estado" que é formado pelos valores armazenados em seus atributos.

Em programação orientada a objetos, tem-se que **atributos** são as características particulares de cada objeto. Tais características são definidas na classe a que o objeto pertence e, dessa forma, todos os objetos dessa classe compartilham das mesmas características, porém, com valores diferentes (CARDOSO, 2006; DEITEL e DEITEL, 2017). Juntamente com os atributos, cada objeto pode realizar operações, as quais são chamadas **métodos**.

> O método armazena as declarações do programa que, na verdade, executam as tarefas; além disso, ele oculta essas declarações do usuário, assim como o pedal do acelerador de um carro oculta do motorista os mecanismos para fazer o veículo ir mais rápido (DEITEL e DEITEL, 2017).

Em síntese, um objeto é uma entidade capaz de reter um estado (através de seus atributos) e que oferece uma série de métodos capazes de examinar ou afetar este estado. Dentro do contexto de execução de um software, para organizá-los, no momento da sua criação lhe é definido um nome ou identificador, exemplo: carro 1 (JOYANES AGUILAR, 2008). Pode-se dizer ainda, que um objeto é um *tipo abstrato de dados*[2], os quais foram definidos pelo programador ao descrever a classe a que tal objeto pertence.

É importante observar que a criação de objetos durante a execução do programa requer o comando de criação, tal comando segue a mesma sintaxe da utilização de variáveis. Todavia, vale ressaltar que a criação de um objeto não se trata apenas de uma declaração de variável, pois o processo necessita de comando específico o qual se denomina **instanciação** (SANTOS 2003).

---

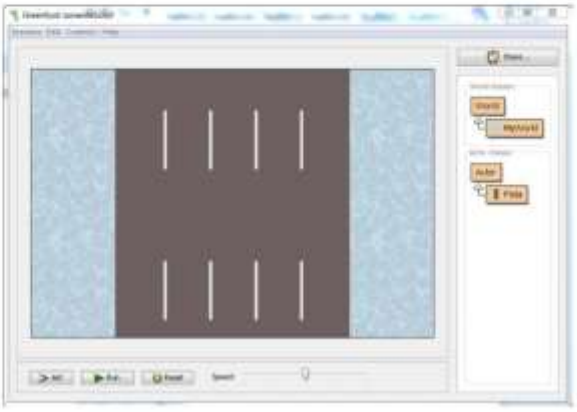
inteiro. Essa associação foi fundada em 1989 e, desde então, fornece padrões OMG direcionados à fornecedores, usuários finais, instituições acadêmicas e agências governamentais (OMG, 2017).

[2] Um tipo abstrato de dados é um invólucro que inclui apenas a representação de dados de um tipo de dados específico e os subprogramas que fornecem as operações para esse tipo. Por meio de controle de acesso, detalhes desnecessários do tipo podem ser ocultados de unidades externas ao invólucro que o usam. Unidades de programa que usam um tipo de dados abstrato podem declarar variáveis de tal tipo, mesmo que a representação real seja deles ocultada. Um exemplar de um tipo de dados abstrato é chamado de um objeto (SEBESTA, 2011, p.505).

Atividade:

Passo 1: Utilizando o Greenfoot, procure construir a interface visualizada na Figura 2.

Figura 2 - Prospecto



Passo 2: Realize a implementação conforme modelagem UML e descrição em anexo.

MyWorld
- pista1: Pista - pista2: Pista
+ MyWorld() + act(): void

Pista
-image: GreenfootImage
+ scroll (int, int): void

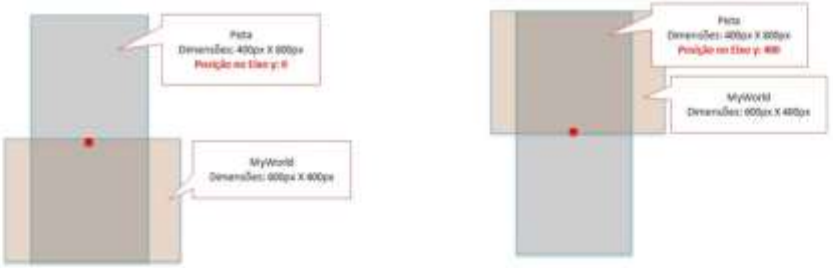
Descrição da classe "Pista": A classe pista tem apenas o atributo 'image' e o método 'scroll'. A 'image' pode ser definida no momento da criação da classe (Figura 3).

Figura 3 - Criando classe Pista

Já o método 'scroll' precisa ser implementado da seguinte maneira:

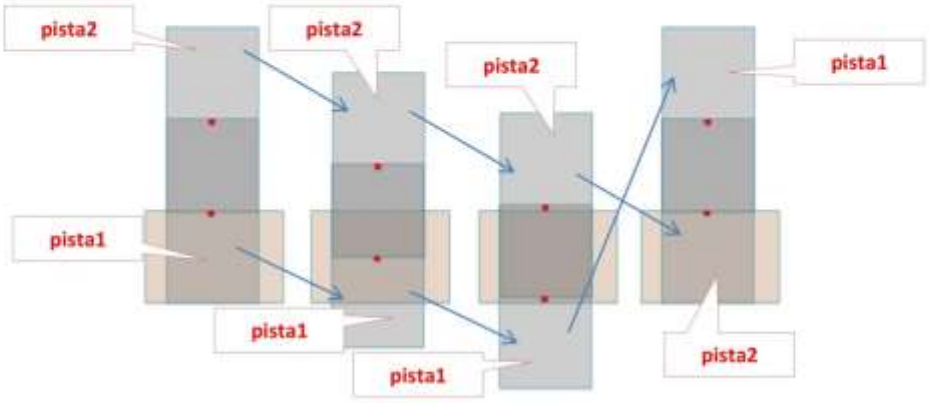
a) A pista tem 800px de altura, enquanto que o cenário (MyWorld) tem 400px. É importante observar que o Greenfoot leva em consideração o meio do objeto para indicar a sua localização no Plano Cartesiano, veja esquema (Figura 4):

Figura 4 - Esquema sobre posicionamento dos objetos no plano cartesiano



b) Vamos utilizar duas "Pistas", sendo que a primeira, "pista1", deve ser criada na posição 0 (zero) do eixo Y de "MyWorld" – em relação ao eixo X, deve ser colocada no centro, ou seja "getWidth()/2". A "pista2" deve ser criada 400px acima da "pista1", ou seja, "-399". Ambas as pistas irão se movimentar no sentido vertical – alterando a posição no eixo Y de acordo com a "velocidade do carro" (a ser implementada posteriormente). Quando a "pista1" chegar na posição 399 do eixo Y, terá sua posição alterada para -399px da "pista2" – o mesmo ocorre com a "pista2" em relação a "pista1" e assim sucessivamente, em loop infinito. Por isso que o método "scroll" tem dois parâmetros, um para capturar a "velocidade do carro" e outro para capturar a posição da pista anterior (veja esquema na Figura 5).

Figura 5 - Funcionamento do método scroll



Descrição da classe MyWorld: A classe MyWorld é formada pelos atributos "pista1" e "pista2". E os métodos "MyWorld()" e "act()". O método construtor "WyWorld()" adiciona os objetos pista1 e pista2 nas coordenadas (300,0) e (300,-399) respectivamente. Já o método "act()", apenas faz a chamada para o método scroll(int, int) para "pista1" e "pista2" em sequência, passando por parâmetro a velocidade (por enquanto estática valorada em '5') e a posição do outro objeto do tipo "Pista" que também está no cenário, diminuída de 399px.

6 Ferramentas

- Software Greenfoot;

7 Avaliação

Questões

1) Fale um pouco sobre as características e ações de cada classe?
2) O que é um método construtor? Quais são as suas características? Descreva onde você implementou essa funcionalidade na classe MyWorld. Qual foi a importância disso para o projeto?
3) O que é envio de mensagem em POO?
4) Em que parte do método scroll você consegue visualizar a existência de troca de mensagens entre objetos?
5) De que forma isso aconteceu?

Referências

CARDOSO, Caíque. **Orientação a objetos na prática:** aprendendo orientação a objetos com Java. Rio de Janeiro: Ciência Moderna, 2006

DEITEL, P.; DEITEL, H. **Java**: como programar. São Paulo: Pearson Education do Brasil, 2017.

JOYANES AGUILAR, Luis. **Fundamentos de programação**: algoritmos, estruturas de dados e objetos. São Paulo: McGraw-Hill, 2008.

OMG. **About OMG**. 2017. Disponível em: < http://www.omg.org/about/index.htm>. Acessado em: 06 nov 2017.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Elsevier, 2003.

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

TUCKER, Allen B.; NOONAN, Robert E. **Linguagens de programação**: princípios e paradigmas. 2.ed. Porto Alegre : AMGH, 2010.

APÊNDICE C – UNIDADE DE ESTUDO 1.3

Aprendendo Orientação a Objetos com Greenfoot

Unidade de Estudo 1.3

(Encapsulamento, Métodos acessores)

1 Objetivos

- Compreender o que é Encapsulamento;
- Entender a importância dos métodos assessores;
- Ratificar os conceitos de classe, método, atributo, instanciação, construtor, envio de mensagem e objeto;

2 Papéis

Estudantes atuam em cooperação, debatendo possibilidades. Professor atua como mediador, lançando proposições e respondendo aos questionamentos dos estudantes.

3 Duração

6h aula;

4 Conteúdos

Anexo 1 – Apresentação POO:
https://drive.google.com/open?id=1Bixy_dKOmVW2DrJdMnRrFZ5cri2ICF96

Anexo 2 – Apostila POO:
https://drive.google.com/open?id=1TOytzDQLnsovZTDdwu7kZlfRf7AefT5c

Anexo 3 - Tutorial Greenfoot:
https://www.greenfoot.org/files/translations/Brazilian/Tutorial%20do%20Greenfoot.htm

Anexo 4 - Banco de imagens:
https://drive.google.com/drive/folders/1zZTYKyKIcAUNMwGyCbDC-fx6_FNnnqgI?usp=sharing

5 Atividades

Leia o texto para resolver a atividade:

Encapsulamento

O encapsulamento é a técnica utilizada para restringir o acesso aos atributos, métodos ou até mesmo à própria classe. Nesse caso, os detalhes da implementação ficam ocultos ao usuário da classe, dessa forma, o usuário passa a utilizar os métodos de uma classe sem se preocupar com detalhes sobre como o método foi implementado internamente (CARVALHO e TEIXEIRA, 2012).

Ocultar aspectos que não precisam ser mostrados ao usuário é caracterizado como um grande trunfo da POO em relação a outros paradigmas de programação,

pois dessa maneira, a complexidade do código permanece transparente para o usuário, de forma que apenas reutiliza a interface previamente implementada.

O encapsulamento de atributos, métodos ou classes se dá por meio da utilização de padrões de acesso, os quais são definidos no memento em que tais elementos são implementados. Em POO podem ser utilizados três formas de padrão de acesso: público; privado e protegido. Tais elementos podem ser descritos da seguinte maneira:

> *public* (público): indica que o método ou o atributo são acessíveis por qualquer classe, ou seja, que podem ser usados por qualquer classe, independentemente de estarem no mesmo pacote ou estarem na mesma hierarquia;
>
> *private* (privado): indica que o método ou o atributo são acessíveis apenas pela própria classe, ou seja, só podem ser utilizados por métodos da própria classe;
>
> *protected* (protegido): indica que o atributo ou o método são acessíveis pela própria classe, por classes do mesmo pacote ou classes da mesma hierarquia [...] (CARVALHO e TEIXEIRA, 2012, grifo no original).

Contudo, o acesso e utilização dos atributos encapsulados de uma classe é possível por meio dos métodos acessores. Tais métodos têm a função de servir de interface de acesso ao conteúdo existente nos atributos da classe. Nesse caso, atributos encapsulados devem ter um método que obtenha o seu valor atual (método *get*) e um método que possibilite alterar o valor do atributo (método *set*) (CARVALHO e TEIXEIRA, 2012). Vejamos o exemplo proposto por (CARVALHO e TEIXEIRA, 2012):

Figura 1 - Classe Conta com atributos encapsulados

```
public class Conta {
    private int numero;
    private String nome_titular;
    private double saldo;

    public void depositar(double valor) {
        this.saldo = this.getSaldo() + valor;
    }

    public boolean sacar(double valor) {
        if (this.getSaldo() >= valor) {
            this.saldo -= valor;
            return true;
        }
        return false;
    }

    public double getSaldo() {
        return saldo;
    }

    public int getNumero() {
        return numero;
    }

    public String getNome_titular() {
        return nome_titular;
    }

    public void setNome_titular(String nome_titular) {
        this.nome_titular = nome_titular;
    }
```

Fonte: Carvalho e Teixeira (2012, p.53).

Por padrão, os atributos "encapsulados" devem ter um método que obtenha o valor atual do atributo (método get) e um método que altere o valor do atributo (método set). Por exemplo, note que na nova versão da classe Conta há um método getNomeTitular() que retorna o nome do titular da conta e um método setNomeTitular(String) que atribui um novo nome ao titular da conta. Mas, como consideramos que o número da conta é atribuído em sua criação (note que os dois construtores da classe exigem o número) e nunca pode ser alterado, criamos apenas o método getNumero(). No caso do saldo, como ele só pode ser alterado por saques e depósitos, não faria sentido criar um método setSaldo. Assim, os métodos depositar e sacar servem para alterar o saldo e o getSaldo() nos retorna o valor atual do saldo (CARVALHO e TEIXEIRA, 2012, p.53-54)..

Atividade

Passo 1: Utilizando o Greenfoot, vamos construir na íntegra o projeto "Simulador" conforme Figura 2 e Vídeo 1 abaixo.

Figura 2 - Imagem Simulador

Vídeo 1 - Vídeo Simulador

https://youtu.be/dIkNHQPIlv0

Passo 2: Realize a implementação conforme modelagem UML e descrição em abaixo.

MODELAGEM DO PROJETO SIMULADOR

Carro
- image: GreenfootImage - velocidadeAtual: double - velocidadeMaxima: int - potencia: float - velocimetro: Velocimetro - volante: Volante - acelerador: Acelerador - freio : Freio
+ act(): void + dirige(): void + acelerar(): void + frear(): void + getVelocímetro(): Velocímetro + getAcelerador(): Acelerador + getFreio(): Freio + getVelocidadeAtual(): double

Freio
- image: GreenfootImage
+ freia(): boolean

Velocímetro
- image: GreenfootImage - ponteiro: Ponteiro
+ atualizaPonteiro(int): void + getPonteiro(): Ponteiro

Acelerador
- image: GreenfootImage
+ acelera(): boolean

Volante
- image: GreenfootImage
+ giraDireita(): boolean + giraEsquerda(): boolean

Ponteiro
-image: GreenfootImage

Pista
-image: GreenfootImage
+ scroll (int, int): void

Mundo
-image: GreenfootImage - pista1: Pista - pista2: Pista - carro: Carro
+ Mundo() + act(): void

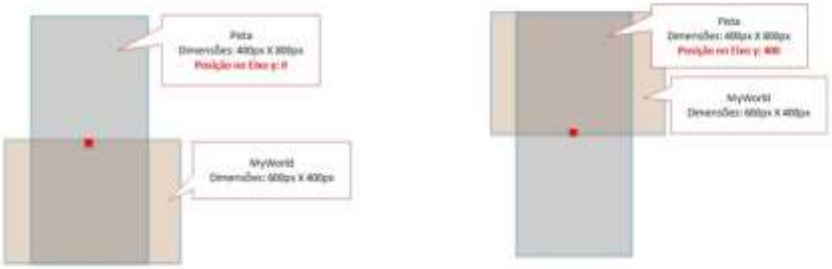
a) Descrição da classe "Pista": A classe pista tem apenas o atributo 'image' e o método 'scroll'. A 'image' pode ser definida no momento da criação da classe (Figura 3).

Figura 3 - Criando classe Pista

Já o método 'scroll' precisa ser implementado da seguinte maneira:

I. A pista tem 800px de altura, enquanto que o cenário (MyWorld) tem 400px. É importante observar que o Greenfoot leva em consideração o meio do objeto para indicar a sua localização no Plano Cartesiano, veja esquema (Figura 4):

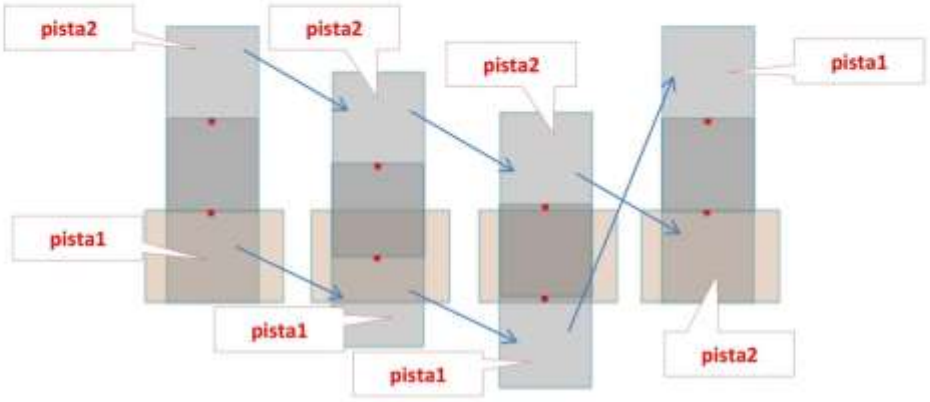
Figura 4 - Esquema sobre posicionamento dos objetos no plano cartesiano



II. Vamos utilizar duas "Pistas", sendo que a primeira, "pista1", deve ser criada na posição 0 (zero) do eixo Y de "MyWorld" – em relação ao eixo X, deve ser colocada no centro, ou seja "getWidth()/2". A "pista2" deve ser criada 399px acima da "pista1", ou seja, "-399". Ambas as pistas irão se movimentar no sentido vertical – alterando a posição no eixo Y de acordo com a "velocidade do carro". Quando a "pista1" chegar na posição 399 do eixo Y, terá sua posição alterada para -399px da "pista2" – o mesmo ocorre com a "pista2" em

relação a "pista1" e assim sucessivamente, em loop infinito. Por isso que o método "scroll" tem dois parâmetros, um para capturar a "velocidade do carr"o e outro para capturar a posição da pista anterior (veja esquema na Figura 5).

Figura 5 - Funcionamento do método scroll



b) **Descrição da classe "Freio":** A classe Freio tem por padrão apenas o atributo "image" que deve ser alimentado com o valor "images/freio.png" que se encontra no banco de imagens. A classe possui o método "freia()" que retorna "*true*", caso a tecla "*down*" seja pressionada – ao mesmo tempo, modifica o atributo "image" para o valor "images/frear.png". Caso a tecla "down" não seja pressionada, retorna "false" e modifica novamente o atributo "image" para o valor padrão ("images/freio.png");

c) **Descrição da classe "Ponteiro":** A classe Ponteiro tem apenas o atributo "image" que deve ser alimentado com o valor "images/ponteiro.png" que se encontra no banco de imagens.

d) **Descrição da classe "Velocimetro":** A classe Velocimetro possui os atributos "image" (GreenfootImage) que deve ser alimentado com o valor "images/ponteiro.png" e ponteiro(Ponteiro); A classe possui também os métodos *atualizaPonteiro*(int) e *getPonteiro*():
   I. *atualizaPonteiro*(int): esse método é do tipo vazio (void) faz a atualização da inclinação (*setRotation*()) do ponteiro de acordo com a velocidade do carro recebida por parâmetro – cada grupo deve

encontrar formas adequadas para sincronizar a posição do ponteiro de acordo com a velocidade do carro.

II. *getPonteiro*(): método que retorna o atributo "ponteiro".

e) **Descrição da classe "Volante":** possui o atributo "image" (GreenfootImage) que deve ser alimentado com o valor "images/volante.png" e os métodos do tipo boleano giraDireita() e giraEsquerda():
   I. *giraDireita*(): retorna "true", caso a tecla "rigth" seja pressionada – ao mesmo tempo que inclina a imagem do objeto para a direita . Caso a tecla "left" não seja pressionada, retorna "false" e modifica a rotação da imagem para 0(zero);
   II. *giraDireita*(): retorna "true", caso a tecla "left" seja pressionada – ao mesmo tempo que inclina a imagem do objeto para a esquerda . Caso a tecla "rigth" não seja pressionada, retorna "false" e modifica a rotação da imagem para 0(zero);

f) **Descrição da classe "Acelerador":** possui apenas o atributo "image" que deve ser alimentado com o valor "images/acelerador.png". A classe possui o método "acelera()" que retorna "*true*", caso a tecla "*up*" seja pressionada – ao mesmo tempo, modifica o atributo "image" para o valor "images/acelerar.png". Caso a tecla "up" não seja pressionada, retorna "false" e modifica novamente o atributo "image" para o valor padrão ("images/acelerador.png");

g) **Descrição da classe Carro:** possui os seguintes atributos e métodos:
   I. **Atributos**
      a. image tipo GreenfootImage - valor padrão "images/car02-n.png";
      b. velocidadeAtual tipo double – valor padrão 1;
      c. velocidadeMaxima tipo int – valor padrão 120;
      d. potencia tipo float – valor padrão 2.0;
      e. velocímetro tipo Velocímetro;
      f. volante tipo Volante;
      g. acelerador tipo Acelerador ;
      h. freio tipo Freio;
   II. **Métodos**:
      a. *act*(): void: ocorre a chamada dos métodos - *acelerar*(), *frear*(), *atualizaPonteiro*((int)) do velocímetro passando por parâmetro o a velocidade atual do carro e o método *dirige*();

b. *dirige*(): void: verifica se o volante foi girado para a esquerda ou para a direita – utiliza os métodos *giraDireita()* e *giraEsquerda()* do volante – modificando a posição no eixo X caso a velocidadeAtual seja maior que 1;
c. *acelerar*(): void: modifica a velocidade do veículo, para isso, se o acelerador for pressionado e a velocidade atual for menor que a velocidade máxima, a velocidade atual é incrementada. Caso o acelerador não seja pressionado, e a velocidade for maior que '1' a velocidade atual é decrementada (carro desacelera);
d. frear(): void: caso o freio seja acionado a velocidadeAtual é decrementada rapidamente;
e. getVelocímetro(): Velocímetro – retorna o velocímetro;
f. getAcelerador(): Acelerador – retorna o acelerador;
g. getFreio(): Freio – retorna o freio;
h. getVelocidadeAtual(): double – retorna a velocidadeatual.

h) **Descrição da classe MyWorld:** A classe MyWorld é formada pelos atributos "pista1", "pista2" e "carro". E os métodos "MyWorld()" e "act()".
   I. O método construtor "*WyWorld*()" adiciona os objetos pista1, pista2 e carro nas coordenadas adequadas – para além disso, adiciona ainda, através dos métodos assessores do carro, os objetos acelerador, freio, volante e velocímetro. E através do velocímetro que está no carro, adiciona o ponteiro (todos em coordenadas adequadas conforme Figura 2).
   II. Já o método "*act*()", apenas faz a chamada para o método scroll(int, int) para "pista1" e "pista2" em sequência, passando por parâmetro a posição do outro objeto do tipo "Pista" que também está no cenário e a velocidadeAtual do carro.

Desafio

A partir do que aprendemos até aqui sobre orientação a objetos e sobre Greenfoot, implemente funcionalidades ao projeto, inserindo outra classe chamada "Obstáculo" os quais devem ir surgindo aleatoriamente sobre a pista, sendo que o veículo deve desviá-los, caso colida: "Game over!".

6 Ferramentas

- Software Greenfoot;

7 Avaliação

Questões

1) O que é encapsulamento?
2) Quais são os modificadores de acesso existentes?
3) Como faço para acessar atributos encapsulados?
4) Em que parte do projeto Simulador você consegue verificar que houve encapsulamento?
5) Se existiu, foi importante para o desenvolvimento do projeto?
6) Sobre os demais conceitos de POO (classe, objeto, instanciação, mensagem, etc), onde você consegue identifica-los no projeto?

Referências

CARVALHO, V. A. de. TEIXEIRA, G. F. **Programação orientada a objetos**: Curso técnico de informática. Colatina: IFES, 2012.

APÊNDICE D – UNIDADE DE ESTUDO 2.1

Aprendendo Orientação a Objetos com Greenfoot

Unidade de Estudo 2.1

(Abstração, Classe, Objeto, Atributo, Método, Construtor, Instanciação, Mensagem)

1 Objetivos

- Compreender o que são objetos computacionais, classes, atributos, métodos e instanciação;
- Entender a relação entre classes e objetos;
- Verificar a importância de construtores no processo de criação de objetos;
- Compreender o funcionamento do envio de mensagens entre objetos.

2 Papéis

Estudantes organizam-se em grupos e professor atua como mediador da proposta.

3 Duração

6h aula;

4 Conteúdos

Anexo 1 – Apresentação POO:
https://drive.google.com/open?id=1Bixy_dKOmVW2DrJdMnRrFZ5cri2ICF96

Anexo 2 – Apostila POO:
https://drive.google.com/open?id=1TOytzDQLnsovZTDdwu7kZlfRf7AeIT5c

Anexo 3 - Tutorial Greenfoot:
https://www.greenfoot.org/files/translations/Brazilian/Tutorial%20do%20Greenfoot.htm

Anexo 4 - Banco de imagens:
https://drive.google.com/open?id=14Jeq-i5YzUZw7csRypQekPzXoDAn8DLS

5 Atividades

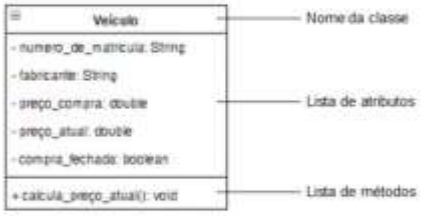
5.1) Leitura dos conceitos fundamentais dessa unidade:

A **abstração** é uma característica fundamental da programação e do projeto orientados a objetos, pois é por meio dela que o projetista a ideia geral da proposta e dos objetivos do produto a ser desenvolvido. Essa característica permite que grandes sistemas sejam especificados em um nível global, antes de ocorrer a implementação dos métodos individuais" (TUCKER e NOONAM, 2010).

Previamente à implementação de uma classe, é realizada a sua modelagem conforme regras estabelecidas pelos padrões de desenvolvimento de software, que

atualmente é realizada pela UML, a qual tem suas especificações padronizadas pela OMG[1]. A Figura 1 ilustra a representação UML de uma classe hipotética 'Veículo'.

Figura 1 – Representação de uma classe



Assim, como se pode verificar na Figura 1, cada objeto criado a partir dessa classe, terá os mesmos atributos e métodos nela descritos. O que será diferente de um objeto em relação a outro será o seu identificador e o seu "estado" que é formado pelos valores armazenados em seus atributos.

No mundo real existem muitos elementos que interagem entre si, sendo que cada um desempenha funções específicas em relação ao seu contexto. Tais elementos são **objetos**. Assim, objetos são "coisas" e podem ser concretos ou abstratos. Um livro, um notebook, uma cadeira ou um veículo são exemplos de objetos concretos comuns do cotidiano das pessoas. De outro lado, uma conta em um banco ou uma equação matemática são exemplos de objetos abstratos que também podem ser implementados em ambiente computacional (CARDOSO, 2006; KEOGH e GIANNINI, 2005; DEITEL e DEITEL, 2017).

Em programação orientada a objetos, tem-se que **atributos** são as características particulares de cada objeto. Tais características são definidas na classe a que o objeto pertence e, dessa forma, todos os objetos dessa classe compartilham das mesmas características, porém, com valores diferentes (CARDOSO, 2006; DEITEL e DEITEL, 2017). Juntamente com os atributos, cada objeto pode realizar operações, as quais são chamadas **métodos**.

> O método armazena as declarações do programa que, na verdade, executam as tarefas; além disso, ele oculta essas declarações do usuário, assim como o pedal do acelerador de um carro oculta do motorista os mecanismos para fazer o veículo ir mais rápido (DEITEL e DEITEL, 2017).

Em síntese, um objeto é uma entidade capaz de reter um estado (através de seus atributos) e que oferece uma série de métodos capazes de examinar ou afetar este estado. Dentro do contexto de execução de um software, para organizá-los, no momento da sua criação lhe é definido um nome ou identificador, exemplo: carro 1 (JOYANES AGUILAR, 2008).

É importante observar que a criação de objetos durante a execução do programa requer o comando de criação, tal comando segue a mesma sintaxe da utilização de

[1] O Object Management Group® (OMG®) é uma associação aberta internacional, sem fins lucrativos, que tem o objetivo de desenvolver padrões internacionais de tecnologia para usuários do mundo inteiro. Essa associação foi fundada em 1989 e, desde então, fornece padrões OMG direcionados à fornecedores, usuários finais, instituições acadêmicas e agências governamentais (OMG, 2017).

variáveis. Todavia, vale ressaltar que a criação de um objeto não se trata apenas de uma declaração de variável, pois o processo necessita de comando específico o qual se denomina **instanciação** (SANTOS 2003). É importante, também, ressaltar que no momento em que um objeto é instanciado, o programador precisa definir quais serão as configurações iniciais desse objeto (estado inicial) – essa tarefa é realizada pela chamada do **construtor** da classe.

Mensagens são a forma de comunicação entre objetos, por meio delas é possível acessar informações retidas no estado de um objeto. As mensagens são responsáveis por ativar os métodos que residem nos objetos, pois são eles – os métodos – que definem como um determinado objeto deve reagir às mensagens a ele enviadas, representando o seu comportamento (DEITEL e DEITEL, 2017).

Já as **Classes** são modelos pelos quais os objetos são criados. É na classe que o desenvolvedor define todas as característica e operações que farão parte dos objetos que dela tiverem origem. Assim, uma classe é uma descrição de um conjunto de objetos, pois nela constam as especificações de atributos e que reúnem características comuns deste conjunto (SANTOS, 2003).

5.2) Atividade Proposta

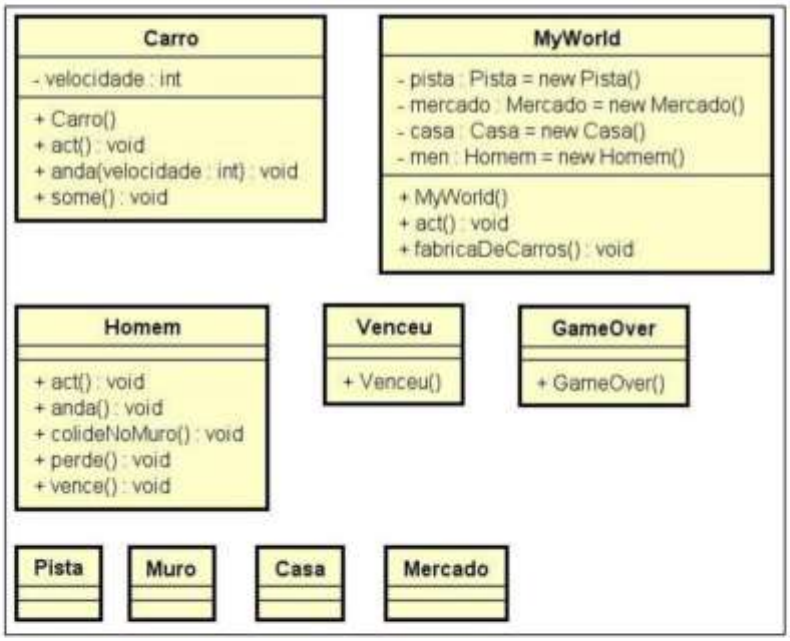
Utilizando o Greenfoot, construir um protótipo de jogo conforme ilustrado pela **Figura 2**, **Vídeo 1** e **Modelagem do Projeto Travessia** (Figura 3). Na sequência responda às questões do item 7 Avaliação.

Figura 2 - Prospecto do Projeto Travessia

Vídeo 1 - Vídeo Projeto Travessia

https://youtu.be/3mkCBIwaCH4

Figura 3 - Diagrama de classes do Projeto Travessia



5.3) Descrição das classes

I. **Descrição da classe MyWorld:** A classe MyWorld é formada pelos seguintes atributos e métodos:

- **Atributos**

a. pista tipo Pista;
b. mercado tipo Mercado;
c. casa tipo Casa;
d. men tipo Homem;

- **Métodos**

a. **MyWorld() (Construtor):** adiciona os objetos *pista*, *mercado*, *casa* e *men* nas coordenadas adequadas – adiciona ainda, os muros nas laterais da lista para impedir que o objeto *men* saia fora da "rodovia", exceto no refúgio reservado para isso. Para inserir os muros, pode-se utilizar laço de repetição para automatizar o processo, sendo que, nesse caso, é necessário incrementar o valor da coordenada 'X', visto que todos os objetos do tipo muro ficaram na coordenada y=50 na parte superior e 450 na parte inferior [`addObject(new Muro(),x, 50); - addObject(new Muro(),x, 450);`].

b. **act():** Esse método é responsável por executar as ações da classe em loop infinito enquanto o jogo estiver rodando, portando todas as ações necessárias à execução do jogo devem ser chamadas nele, porém nesse caso, é necessário apenas chamar o método *fabricaDeCarros();*

c. **fabricaDeCarros():** esse método tem por objetivo distribuir aleatoriamente carros sobre a pista, para isso é necessário sortear a coordenada y para inserir um obstáculo visto que a coordenada x pode ser sempre igual a 0 (zero). Nesse caso, utiliza-se o comando `int y = Greenfoot.getRandomNumber(int alcance);` sendo que o valor máximo para y deve estar entre 85 e 410 que é a extensão da 'pista' do cenário. Dica: se atribuir o alcance de apenas 410 para o método getRandomNumber(int), muitos objetos serão inseridos, inviabilizando a execução. Nesse caso pode-se atribuir um valor significativamente maior (ex: 5000) e selecionar (if) apenas aqueles que forem menores que 410 e maiores que 85 – outra coisa que deve-se fazer para diminuir o numero de itens sorteados é selecionar apenas aqueles que forem divisíveis por determinado número (ex: y%6==0).

II. **Descrição da classe Carro:** A classe Carro é formada pelos seguintes atributos e métodos:

- **Atributos**

a. velocidade tipo int;

- **Métodos**

a. **Carro() (Construtor):** inicializa os atributos *image* e *velocidade* do carro, nesse caso, realiza o sorteio aleatório de um valor que vai de 1 a 6 (pois temos seis tipos de carros) [`int x = 1+Greenfoot.getRandomNumber(6);`]. A imagem é adicionada ao objeto pelo comando setImage(String) [`setImage(x+".png");`] e velocidade do objeto vai ser o dobro de x [`velocidade = x*2;`];

b. **act():**Esse método é responsável por executar as ações da classe em loop infinito enquanto o jogo estiver rodando, portando todas as ações necessárias à execução do jogo devem ser chamadas nele, porém nesse

caso, é necessário apenas chamar o método anda(int velocidade), passando por parâmetro a velocidade do objeto [`this.velocidade`];

c. **anda(int velocidade):** método responsável por fazer os carros se movimentarem no sentido horizontal (da esquerda para a direita), de acordo com a sua velocidade. Utiliza-se o método herdado de Actor move(int), passando por parâmetro a velocidade do objeto [`this.velocidade`];

d. **some():** verifica se o objeto chegou ao final do cenário [isAtEdge()] e, caso afirmativo, remove-o do "mundo" [getWorld.removeObject(this)];

III. **Descrição da classe Homem:** A classe Homem não possui atributos próprios, apenas os herdados de Actor. Mas contém os seguintes métodos:

➢ **Métodos**

a. **act():** Esse método é responsável por executar as ações da classe em loop infinito enquanto o jogo estiver rodando, portando todas as ações necessárias à execução do jogo devem ser chamadas nele, nesse caso, é necessário chamar os métodos *anda(), colideNoMuro(), perde()* e *vence();*

b. **anda():** método responsável por fazer os objeto se movimentar conforme o jogador pressiona as teclas "up", "down", "left" e "right" do teclado. Para verificar se a tecla foi pressionada utiliza-se o método isKeyDown(String) da classe Greenfoot [`isKeyDown("up")`]. Utiliza-se o método herdado de Actor setLocation(x,y), para realizar os movimentos, sendo que o movimento vertical se dá pelo incremento/decremento da posição na coordenada y e o movimento horizontal acontece pela variação do valor da coordenada x [`setLocation(getX(), getY()-1);`] – [`setLocation(getX()+1, getY());`]. Para dar melhor qualidade visual pode-se alterar a imagem do objeto (setImage(String)) quando estiver se deslocando nos sentidos esquerda e direita, alternando entre as imagens "*menD.png*" e "*menE.png*";

c. **colideNoMuro():** Esse método tem a finalidade de impedir que o objeto *men* do tipo *Homem* possa passar por cima dos objetos do tipo muro, nesse caso, utiliza-se o método da classe Actor getOneIntersectingObject(class<?>) para verificar se o objeto *men* interceptou outo objeto da classe Muro, caso verdadeiro, verificar de está abaixo ou acima dele, nesse caso modificar a posição do objeto *men*(em 40px) para que não consiga atravessar o muro.

```
public void colideNoMuro(){
    Actor a = getOneIntersectingObject(Muro.class);
    if(a!=null){
    if(a.getY()<this.getY()){
        setLocation(getX(), a.getY()+40);
    }else{
        setLocation(getX(), a.getY()-40);
    }
}
}
```

d. **perde():** verifica se o objeto *men* colidir com uma instância da classe Carro, um objeto da classe GameOver é instanciado na coordenada x=500 e y=250 e o jogo é encerrado [Greenfoot.stop()] – a exemplo do

método colideNoMuro(), utiliza-se o método *getOneIntersectingObject(class<?>)* [`Actor a=getOneIntersectingObject(Carro.class);`];

e. **vence():** verifica se o objeto *men* intercepta uma instância da classe Mercado e, caso isso aconteça, um objeto da classe Venceu é instanciado na coordenada x=500 e y=250 e o jogo é encerrado [Greenfoot.stop()] – a exemplo dos métodos colideNoMuro() e perde(), utiliza-se o método *getOneIntersectingObject(class<?>)* [`Actor a=getOneIntersectingObject(Mercado.class);`].

IV. **Descrição da classe GameOver:** não possui atributos. Tem a finalidade de criar objeto contendo o texto que indica derrota do personagem principal. Tem apenas o método construtor:

a. **GameOver() (Construtor):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "Game over", com tamanho 60px, na cor vermelha, fundo amarelo e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [setImage(GreenfootImage)].

```
public GameOver(){
    GreenfootImage texto;
    texto = new GreenfootImage("Game Over", 60, Color.RED, new Color(0,0,0,0) , Color.BLACK);
    setImage(texto);
}
```

V. **Descrição da classe Venceu:** Tem a finalidade de criar objeto contendo o texto que indica vitória do personagem principal. Tem apenas o método construtor:

a. **Venceu() (Construtor):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "Você venceu!!", com tamanho 60px, na cor vermelha, fundo amarelo e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [setImage(GreenfootImage)].

As classes *"Pista"*, *"Muro"*, *"Casa"* e "*Mercado*" não possuem atributos e métodos além dos herdados de Actor. O único detalhe dessas classes é definir adequadamente as suas respectivas imagens: *"pista.png"*, *"muro.png"*, *"house-5.png"* e *"mercado1.png"*. isso é possível fazer no momento da sua criação, conforme ilustração ao lado. Para que as imagens estejam disponíveis conforme visualização ao lado, é necessário baixa-las do link Anexo 4 e armazená-las na pasta "images" do projeto.

Desafio

Vamos nos desafiar no processo de aprendizagem de POO. Pense em alguma funcionalidade extra para facilitar a travessia do objeto *men*. Como sugestão, podemos pensar em algum objeto que lhe dê maior velocidade, ou em um refúgio na área central da pista, onde os carros precisam desviar. Use sua criatividade.

6 Ferramentas

- Software Greenfoot;

7 Avaliação

7.1 Questões

1) Analisando a figura e vídeo, quais objetos podemos identificar?
2) Em relação à POO, o que são objetos?
3) O que são atributos? Ao desenvolver o projeto "Travessia", quais atributos foram possíveis visualizar?
4) Os objetos precisam ser modelados (descritos) – "*antes de poder dirigir um carro, alguém tem de projetá-lo*"(DEITEL & DEITEL, 2016, p.8) - para que sejam determinadas as suas características e funcionalidades dentro do sistema. Assim, muitos objetos pertencem a um mesmo "grupo". Em POO, como esse grupo é denominado?
5) Ao estudarmos POO, aprendemos que a primeira fase do desenvolvimento de um software sob a abordagem de POO, inicialmente construímos a modelagem das classes a serem implementadas. Que nome é atribuído a esse processo? O que é importante observar nesse processo?
6) Ao analisar o vídeo, quais objetos exercem algum tipo de ação? Descreva de maneira literal essas ações.
7) Construtores são importantes para "definir quais serão as configurações iniciais do objeto (estado inicial)". Em que ponto do desenvolvimento do projeto essa afirmação se é confirmada?
8) Para utilizar objetos é preciso cria-los. Depois de implementar o projeto Travessia, indique onde foi necessário instanciar objetos. Como isso foi realizado?
9) Objetos se comunicam entre si por meio do envio e recebimento de mensagens, em que ponto do desenvolvimento do projeto "Travessia" isso fica evidente? Como isso acontece?

Referências

CARDOSO, Caíque. **Orientação a objetos na prática:** aprendendo orientação a objetos com Java. Rio de Janeiro: Ciência Moderna, 2006

DEITEL, P.; DEITEL, H. **Java**: como programar. São Paulo: Pearson Education do Brasil, 2017.

JOYANES AGUILAR, Luis. **Fundamentos de programação**: algoritmos, estruturas de dados e objetos. São Paulo: McGraw-Hill, 2008.

KEOGH, James Edward; GIANNINI, Mario. **OOP desmitificado:** programação orientada a objetos. Rio de Janeiro: Alta Books, 2005.

OMG. **About OMG**. 2017. Disponível em: < http://www.omg.org/about/index.htm>. Acessado em: 06 nov 2017.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Elsevier, 2003.

TUCKER, Allen B.; NOONAN, Robert E. **Linguagens de programação**: princípios e paradigmas. 2.ed. Porto Alegre : AMGH, 2010.

APÊNDICE E – UNIDADE DE ESTUDO 2.2

Aprendendo Orientação a Objetos com Greenfoot

Unidade de Estudo 2.2

(Encapsulamento, Métodos acessores)

1 Objetivos

- Compreender o que é Encapsulamento;
- Entender a importância dos métodos assessores;
- Ratificar os conceitos de classe, método, atributo, instanciação, construtor, envio de mensagem e objeto;

2 Papéis

Estudantes atuam em cooperação, debatendo possibilidades. Professor atua como mediador, lançando proposições e respondendo aos questionamentos dos estudantes.

3 Duração

9h aula;

4 Conteúdos

Anexo 1 – Apresentação POO:
https://drive.google.com/open?id=1Bixy_dKOmVW2DrJdMnRrFZ5cri2ICF96

Anexo 2 – Apostila POO:
https://drive.google.com/open?id=1TOytzDQLnsovZTDdwu7kZlfRf7AeIT5c

Anexo 3 - Tutorial Greenfoot:
https://www.greenfoot.org/files/translations/Brazilian/Tutorial%20do%20Greenfoot.htm

Anexo 4 - Banco de imagens:
https://drive.google.com/drive/folders/1paKGvsUX8Q591MVW2SX3qbErdFpyejrN?usp=sharing

5 Atividades

5.1) Leitura dos conceitos fundamentais dessa unidade:

Encapsulamento

O encapsulamento é a técnica utilizada para restringir o acesso aos atributos, métodos ou até mesmo à própria classe. Nesse caso, os detalhes da implementação ficam ocultos ao usuário da classe, dessa forma, o usuário passa a utilizar os métodos de uma classe sem se preocupar com detalhes sobre como o método foi implementado internamente (CARVALHO e TEIXEIRA, 2012).

Ocultar aspectos que não precisam ser mostrados ao usuário é caracterizado como um grande trunfo da POO em relação a outros paradigmas de programação,

pois dessa maneira, a complexidade do código permanece transparente para o usuário, de forma que apenas reutiliza a interface previamente implementada.

O encapsulamento de atributos, métodos ou classes se dá por meio da utilização de padrões de acesso, os quais são definidos no memento em que tais elementos são implementados. Em POO podem ser utilizados três formas de padrão de acesso: público; privado e protegido. Tais elementos podem ser descritos da seguinte maneira:

> *public* (público): indica que o método ou o atributo são acessíveis por qualquer classe, ou seja, que podem ser usados por qualquer classe, independentemente de estarem no mesmo pacote ou estarem na mesma hierarquia;
>
> *private* (privado): indica que o método ou o atributo são acessíveis apenas pela própria classe, ou seja, só podem ser utilizados por métodos da própria classe;
>
> *protected* (protegido): indica que o atributo ou o método são acessíveis pela própria classe, por classes do mesmo pacote ou classes da mesma hierarquia [...] (CARVALHO e TEIXEIRA, 2012, grifo no original).

Contudo, o acesso e utilização dos atributos encapsulados de uma classe é possível por meio dos métodos acessores. Tais métodos têm a função de servir de interface de acesso ao conteúdo existente nos atributos da classe. Nesse caso, atributos encapsulados devem ter um método que obtenha o seu valor atual (método *get*) e um método que possibilite alterar o valor do atributo (método *set*) (CARVALHO e TEIXEIRA, 2012). Vejamos o exemplo proposto por (CARVALHO e TEIXEIRA, 2012):

Figura 1 - Classe Conta com atributos encapsulados

```
public class Conta {
    private int numero;
    private String nome_titular;
    private double saldo;

    public void depositar(double valor) {
        this.saldo = this.getSaldo() + valor;
    }

    public boolean sacar(double valor) {
        if (this.getSaldo() >= valor) {
            this.saldo -= valor;
            return true;
        }
        return false;
    }

    public double getSaldo() {
        return saldo;
    }

    public int getNumero() {
        return numero;
    }

    public String getNome_titular() {
        return nome_titular;
    }

    public void setNome_titular(String nome_titular) {
        this.nome_titular = nome_titular;
    }
```

Fonte: Carvalho e Teixeira (2012, p.53).

Por padrão, os atributos "encapsulados" devem ter um método que obtenha o valor atual do atributo (método get) e um método que altere o valor do atributo (método set). Por exemplo, note que na nova versão da classe Conta há um método getNomeTitular() que retorna o nome do titular da conta e um método setNomeTitular(String) que atribui um novo nome ao titular da conta. Mas, como consideramos que o número da conta é atribuído em sua criação (note que os dois construtores da classe exigem o número) e nunca pode ser alterado, criamos apenas o método getNumero(). No caso do saldo, como ele só pode ser alterado por saques e depósitos, não faria sentido criar um método setSaldo. Assim, os métodos depositar e sacar servem para alterar o saldo e o getSaldo() nos retorna o valor atual do saldo (CARVALHO e TEIXEIRA, 2012, p.53-54)..

5.2) Atividade Proposta

Passo 1: Utilizando o Greenfoot, vamos construir na íntegra o projeto "Simulador" conforme Figura 2 e Vídeo 1 abaixo.

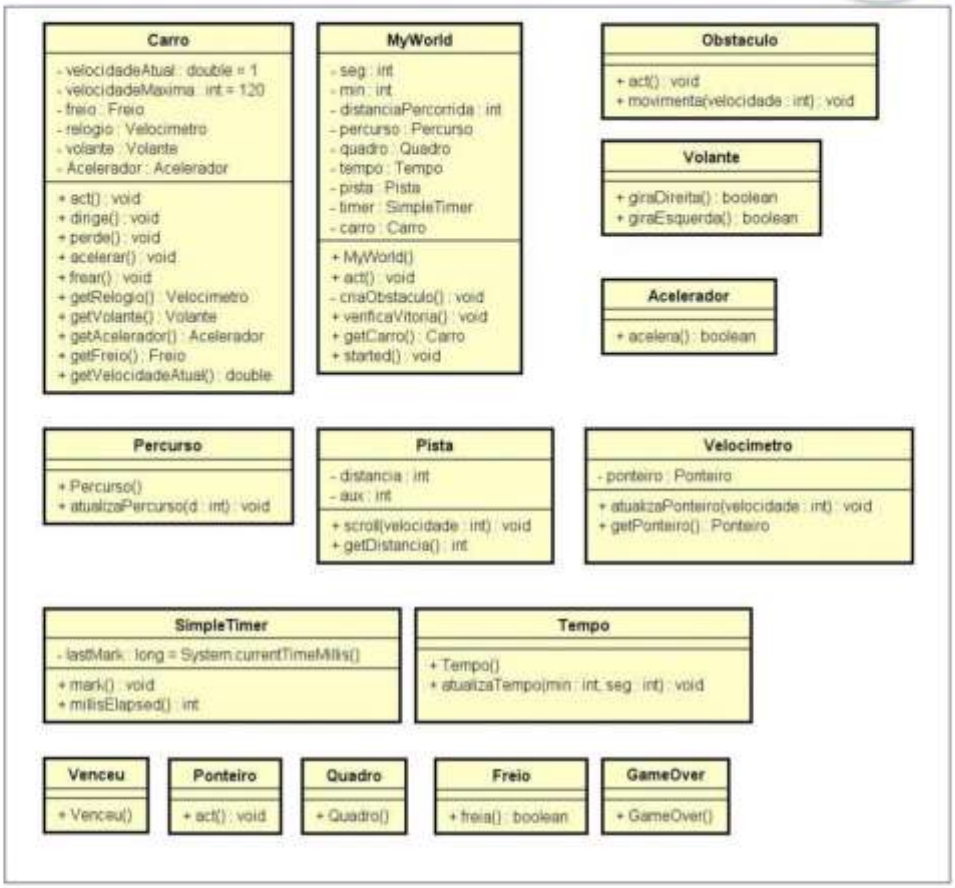
Figura 2 - Imagem Simulador

Vídeo 1 - Vídeo Simulador

https://youtu.be/SXUI2m1Rfy4

Passo 2: Realize a implementação conforme modelagem UML e descrição em abaixo.

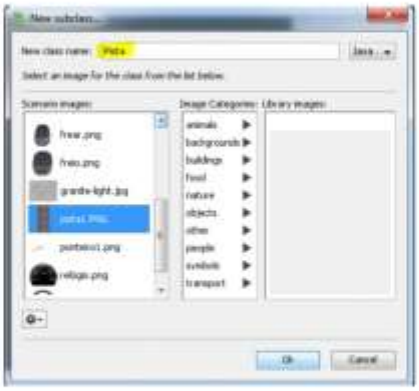
MODELAGEM DO PROJETO SIMULADOR



5.3) Descrição das classes

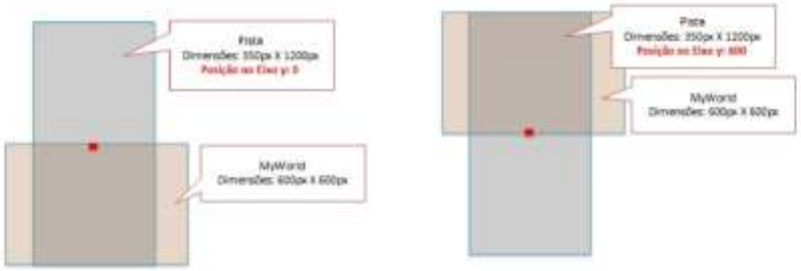
I. **Descrição da classe "Pista":** A classe pista tem apenas os atributos 'distancia' e 'aux' e os métodos 'scroll()' e 'getDistancia()'. A 'image' pode ser definida no momento da criação da classe (Figura 3).

Figura 3 - Criando classe Pista



a. scroll(): A pista tem 1200px de altura, enquanto que o cenário (MyWorld) tem 600px. É importante observar que o Greenfoot leva em consideração o meio do objeto para indicar a sua localização no Plano Cartesiano, veja esquema (Figura 4):

Figura 4 - Esquema sobre posicionamento dos objetos no plano cartesiano



A pista deve ser criada na posição 0 (zero) do eixo Y de "MyWorld" e, em relação ao eixo X, deve ser colocada no centro, ou seja "getWidth()/2". A pista deve deslizar no sentido vertical – alterando a posição no eixo Y de acordo com a "velocidade do carro". Quando chegar na posição 600 do eixo Y, terá sua posição alterada para 0(zero) do eixo Y, mantendo a mesma posição na coordenada X e assim sucessivamente em loop infinito. A velocidade do movimento é dada pela velocidade do carro (passada por parâmetro). (veja esquema na Figura 5).

b. **getDistancia():** retorna o atributo distancia.

II. **Descrição da classe "Freio":** A classe Freio tem por padrão apenas o atributo "image" que deve ser alimentado com o valor "images/freio.png" que se encontra no banco de imagens.

a. **freia():** retorna "*true*", caso a tecla "*down*" seja pressionada – ao mesmo tempo, modifica o atributo "image" para o valor "images/frear.png". Caso a tecla "down" não seja pressionada, retorna "false" e modifica novamente o atributo "image" para o valor padrão ("images/freio.png");

b. **Descrição da classe "Ponteiro":** A classe Ponteiro tem apenas o atributo "image" que deve ser alimentado com o valor "images/ponteiro.png" que se encontra no banco de imagens.

III. **Descrição da classe "Velocimetro":** A classe Velocimetro possui os atributos "image" (GreenfootImage) que deve ser alimentado com o valor "images/ponteiro.png" e ponteiro. A classe possui também os métodos *atualizaPonteiro*(int) e *getPonteiro*():

a. *atualizaPonteiro*(int): esse método é do tipo vazio (*void*) faz a atualização da inclinação (*setRotation*()) do ponteiro de acordo com a velocidade do carro recebida por parâmetro – cada grupo deve encontrar formas adequadas para sincronizar a posição do ponteiro de acordo com a velocidade do carro.

b. *getPonteiro*(): método que retorna o atributo "ponteiro".

IV. **Descrição da classe "Volante":** possui o atributo "image" (GreenfootImage) que deve ser alimentado com o valor "images/volante.png" e os métodos do tipo boleano giraDireita() e giraEsquerda():

a. *giraDireita*(): retorna "true", caso a tecla "rigth" seja pressionada – ao mesmo tempo que inclina a imagem do objeto para a direita (40º) . Caso a tecla "left" não seja pressionada, retorna "false" e modifica a rotação da imagem para 0(zero);

b. *giraDireita*(): retorna "true", caso a tecla "left" seja pressionada – ao mesmo tempo que inclina a imagem do objeto para a esquerda (-40°) . Caso a tecla "rigth" não seja pressionada, retorna "false" e modifica a rotação da imagem para 0(zero);

V. **Descrição da classe "Acelerador":** possui apenas o atributo "image" que deve ser alimentado com o valor "images/acelerador.png". A classe possui o método "acelera()" que retorna "*true*", caso a tecla "*up*" seja pressionada – ao mesmo tempo, modifica o atributo "image" para o valor "images/acelerar.png". Caso a tecla "up" não seja pressionada, retorna "false" e modifica novamente o atributo "image" para o valor padrão ("images/acelerador.png");

VI. **Descrição da classe Carro:** possui os seguintes atributos e métodos:

- **Atributos**

a. image tipo GreenfootImage - valor padrão "images/car02-n.png";
b. velocidadeAtual tipo double – valor padrão 1;
c. velocidadeMaxima tipo int – valor padrão 120;
d. velocimetro tipo Velocimetro;
e. volante tipo Volante;
f. acelerador tipo Acelerador ;
g. freio tipo Freio;

- **Métodos**:

a. ***act*():** ocorre a chamada dos métodos - *acelerar*(), *frear*(), *atualizaPonteiro*((int)) do velocímetro passando por parâmetro o a velocidade atual do carro, o método *dirige*() e *perde();*
b. ***dirige*():** verifica se o volante foi girado para a esquerda ou para a direita – utiliza os métodos *giraDireita()* e *giraEsquerda()* do volante – modificando a posição no eixo X caso a velocidadeAtual seja maior que 1;
c. ***acelerar*():** modifica a velocidade do veículo, para isso, se o acelerador for pressionado e a velocidade atual for menor que a velocidade máxima, a velocidade atual é incrementada. Caso o acelerador não seja pressionado, e

a velocidade for maior que '1' a velocidade atual é decrementada (carro desacelera);

d. **frear():** caso o freio seja acionado a velocidadeAtual é decrementada rapidamente (-3);
e. **perde():** verifica se o objeto colidir com uma instância da classe Obstáculo [getOneIntersectingObject(Obstaculo.class)], um objeto da classe GameOver é instanciado na coordenada x=300 e y=300 e o jogo é encerrado [Greenfoot.stop()];
f. **getVelocímetro():** retorna o velocímetro;
g. **getAcelerador():** retorna o acelerador;
h. **getVolante():** retorna o volante;
i. **getFreio():** retorna o freio;
j. **getVelocidadeAtual():** retorna a velocidadeAtual.

VII. **Descrição da classe MyWorld:** A classe MyWorld é formada pelos seguintes atributos e métodos:

➢ **Atributos**

a. seg tipo inteiro;
b. min tipo inteiro;
c. distanciaPercorrida tipo inteiro;
d. percurso tipo Percurso;
e. quadro tipo Quadro;
f. tempo tipo Tempo;
g. pista tipo Pista;
h. timer tipo SimpleTimer;
i. carro tipo Carro.

b. **Métodos**

a. **MyWorld() (construtor):** adiciona os objetos pista, quadro, tempo, percurso e carro nas coordenadas adequadas – para além disso, adiciona ainda, através dos métodos assessores do carro, os objetos acelerador, freio, volante e velocímetro. E através do velocímetro que está no carro, adiciona o ponteiro (todos em coordenadas adequadas conforme Figura 2).
b. **act():** faz a chamada para o método scroll(int) do atributo "pista", passando por parâmetro a velocidadeAtual do carro [*carro.getVelocidadeAtual()*]. Ainda, faz a chamada dos métodos criaObstáculo() e verificaVitoria(). Também é nesse método que o tempo decorrido é atualizado [*tempo.atualizaTempo(min,seg)*], para isso, utiliza os métodos do atributo timer [*mark()* e *millisElapsed()*] – nesse caso, os segundos [seg] são obtidos pelo comando *seg = millisElapsed()/1000;* Já os minutos [min] são incrementados cada vez que *millisElapsed()* ultrapassar 60000 – nesse momento o timer é reiniciado [*timer.mark()*];

c. **criaObstáculo():** esse método tem por objetivo distribuir aleatoriamente obstáculos sobre a pista, para isso é necessário sortear a coordenada x para inserir um obstáculo visto que a coordenada y pode ser sempre igual a 0 (zero). Nesse caso, utiliza-se o comando int x = Greenfoot.getRandomNumber(int alcance); sendo que o valor máximo para x deve estar entre 140 e 460 que é a extensão da 'pista' do cenário. Dica: se atribuir o alcance de apenas 460 para o método getRandomNumber, muitos objetos serão inseridos, inviabilizando a execução, nesse caso pode-se atribuir um valor significativamente maior (ex: 10000) e selecionar (if) apenas aqueles que forem menores que 460 e maiores que 140 – outra coisa que deve-se fazer para diminuir o numero de itens sorteados é selecionar apenas aqueles que forem divisíveis por determinado número (ex: x%6==0). Ainda, obstáculos somente devem ser sorteados se a velocidadeAtual do 'carro' for maior que 2.
d. **verificaVitoria():** verifica se o atributo distanciaPercorrida é maior ou igual a 5000 – caso a expressão seja verdadeira um objeto do tipo Venceu é instanciado na coordenada x=300 e y=300 do cenário e o jogo é finalizado [*Greenfoot.stop()*].
e. **getCarro():** retorna o atributo carro;
f. **started():** esse método é herdado da classe Greenfoot e para esse jogo, tem a função de inicializar a contagem de tempo, pela chamada do método mark() para o objeto timer [timer.mark()]. O método started() é chamado uma única vez ao início da execução de um jogo no ambiente Greenfoot – nesse caso, serve de "gatilho" para iniciar a contagem de tempo.

VIII. **Descrição da classe Obstaculo:** A classe Obstaculo é formada pelos seguintes métodos:
a. **movimenta(int):** os obstáculos devem ser inseridos no cenário devem deslizar no sentido vertical – alterando a posição no eixo Y sendo que a velocidade do movimento é dada pela velocidade do carro dividida por 5 (passada por parâmetro).
b. **act():** apenas ocorre a chamada do método movimenta, passando por parâmetro a velocidadeAtual do carro que está no MyWorld [*movimenta((int)((MyWorld)getWorld()).getCarro().getVelocidadeAtual());*]

IX. **Descrição da classe SimpleTimer:** Essa classe é nativa do Greenfoot, precisa ser importada para dentro do projeto – isso é possível pelo menu Edit >Import class... Tem a finalidade de contabilizar o tempo decorrido em milissegundos. Não é necessário alterá-la, pois já tem o atributo lastMark, que armazena o tempo decorrido e os métodos mark() e millisElapsed().
a. **mark():** reinicia o valor da variável lastMark;

b. **millisElapsed ():** retorna o tempo decorrido que está armazenado no atributo lastMark;

X. **Descrição da classe Percurso:** não possui atributos. Tem a finalidade de criar objetos que apresentam informações textuais no cenário.

a. **Percurso()(construtor):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "Percurso:", com tamanho 20px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

b. **atualizaPercurso(int):** método que tema função de alterar o texto exibido pelo objeto, sendo que a distância percorrida é passada por parâmetro e juntada com o texto padrão para atualizar a imagem. Todavia é necessário criar um objeto do tipo GreenfootImage a partir do valor informado com tamanho 20px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

XI. **Descrição da classe Tempo:** *Extends* de Actor – não possui atributos. Tem a finalidade de criar objetos que apresentam informações textuais no cenário.

a. **Tempo()(construtor):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "Tempo:", com tamanho 20px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

b. **atualizaTempo(int, int):** método que tema função de alterar o texto exibido pelo objeto, sendo que os minutos e segundos são passados por parâmetro e juntada com o texto padrão para atualizar a imagem. Todavia é necessário criar um objeto do tipo GreenfootImage a partir dos valores informados com tamanho 20px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

XII. **Descrição da classe GameOver:** não possui atributos. Tem a finalidade de criar objeto contendo o texto que indica derrota do personagem principal.

a. **GameOver()(construtor):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "game Over", com tamanho 60px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

XIII. **Descrição da classe Venceu:** não possui atributos. Tem a finalidade de criar objeto contendo o texto que indica vitória do personagem principal.

a. **Venceu():** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "Você Venceu!!!", com tamanho 60px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

XIV. **Descrição da classe Quadro:** não possui atributos. Tem a finalidade de criar um quadro branco com as bordas pretas para que os objetos de tempo de percurso sejam melhor visualizados.

a. **Quadro():** Instancia um objeto do tipo GreenfootImage sem texto, pois tem a finalidade de montar um quadro com as dimensões 125 X 100. Para desenhar as bordas pretas deve-se utilizar o comando setColor() [setColor(Color.BLACK);] e na sequencia executar o comando set drawRect(0, 0, quadro.getWidth()-1, quadro.getHeight()-1); para construir as bordas. Na sequencia muda-se a cor para branco [Color(Color.WHITE);] para preencher o fundo com a cor branca, aplica-se o comando fillRect(1, 1, quadro.getWidth()-2, quadro.getHeight()-2);. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

Desafio

A partir do que aprendemos até aqui sobre orientação a objetos e sobre Greenfoot, implemente funcionalidades ao projeto, inserindo outra classe chamada "SuperObstaculo" que extende de Obstaculo – sendo que os objetos que dela tiverem origem devem, além de deslizar na vertical movimentar-se lateralmente de maneira aleatória, a fim de dificultar ainda mais o percurso do carro. Implemente também, um cálculo da velocidade média alcançada pelo veículo quando vencer o jogo (levar em consideração o tempo e o espaço percorrido) – o valor da velocidade média deve ser exibido junto da mensagem de vitória.

6 Ferramentas

- Software Greenfoot;

7 Avaliação

Questões

1) O que é encapsulamento?
2) Quais são os modificadores de acesso existentes?
3) Como faço para acessar atributos encapsulados?
4) Em que parte do projeto Simulador você consegue verificar que houve encapsulamento?

5) Se existiu, foi importante para o desenvolvimento do projeto?
6) Sobre os demais conceitos de POO (classe, objeto, instanciação, mensagem, etc), onde você consegue identifica-los no projeto?

Referências

CARVALHO, V. A. de. TEIXEIRA, G. F. **Programação orientada a objetos**: Curso técnico de informática. Colatina: IFES, 2012.

APÊNDICE F – UNIDADE DE ESTUDO 2.3

Aprendendo Orientação a Objetos com Greenfoot

Unidade de Estudo 2.3

{União de todos os conceitos}

1 Objetivos

- Ratificar os conceitos de classe, objeto, método, atributo, instanciação, construtor, envio de mensagem, encapsulamento, herança e polimorfismo;
- Perceber como os conceitos de PPO se relacionam a fim de possibilitar a construção eficiente de um produto de software;

2 Papéis

Estudantes atuam em cooperação, debatendo possibilidades. Professor atua como mediador, lançando proposições e respondendo aos questionamentos dos estudantes.

3 Duração

9h aula;

4 Conteúdos

Anexo 1 – Apresentação POO:
https://drive.google.com/open?id=1Bixy_dKOmVW2DrJdMnRrFZ5cri2ICF96

Anexo 2 – Apostila POO:
https://drive.google.com/open?id=1TOytzDQLnsovZTDdwu7kZlfRf7AeIT5c

Anexo 3 - Tutorial Greenfoot:
https://www.greenfoot.org/files/translations/Brazilian/Tutorial%20do%20Greenfoot.htm

Anexo 4 - Banco de imagens e sons:
https://drive.google.com/drive/folders/1_He062HnfW2Csv0-cBvkXm6UQhG-6nlg?usp=sharing

Anexo 5 – Vídeo 1: https://youtu.be/6RPII9kGh90

5 Atividades

5.1 Considere as premissas básicas relacionadas à Programação Orientada a Objetos:

Aprendendo Orientação a Objetos com Greenfoot

Premissa 1

"O mundo é formado por objetos."

No mundo real existem muitos elementos que interagem entre si, sendo que cada um desempenha funções específicas em relação ao seu contexto. Tais elementos são objetos. Assim, objetos são "coisas" e podem ser concretos ou abstratos. Um livro, um notebook, uma cadeira ou um veículo são exemplos de objetos concretos comuns do cotidiano das pessoas. De outro lado, uma conta em um banco ou uma equação matemática são exemplos de objetos abstratos que também podem ser implementados em ambiente computacional (CARDOSO, 2006; KEOGH e GIANNINI, 2005; DEITEL e DEITEL, 2017).

Premissa 2

"Objeto é constituído pela definição de atributos e métodos."

Em programação orientada a objetos, tem-se que ***atributos*** *são as características particulares de cada objeto. Tais características são definidas na classe a que o objeto pertence e, dessa forma, todos os objetos dessa classe compartilham das mesmas características, porém, com valores diferentes(CARDOSO, 2006; DEITEL e DEITEL, 2017). Juntamente com os atributos, cada objeto pode realizar operações, as quais são chamadas* ***métodos****.*

> *O método armazena as declarações do programa que, na verdade, executam as tarefas; além disso, ele oculta essas declarações do usuário, assim como o pedal do acelerador de um carro oculta do motorista os mecanismos para fazer o veículo ir mais rápido (DEITEL e DEITEL, 2017).*

Em síntese, um objeto é uma entidade capaz de reter um estado (através de seus atributos) e que oferece uma série de métodos capazes de examinar ou afetar este estado.

Premissa 3

"Objetos são organizados em classes."

Classes são modelos pelos quais os objetos são descritos. É na classe que o desenvolvedor define todas as característica e operações que farão parte dos objetos que dela tiverem origem. Assim, uma classe é uma descrição de um conjunto de objetos, pois nela constam as especificações de atributos e que reúnem características comuns deste conjunto (SANTOS, 2003).

Premissa 4

"Objetos se comunicam por meio do envio de mensagens."

Mensagens são a forma de comunicação entre objetos, por meio delas é possível acessar informações retidas no estado de um objeto. As mensagens são responsáveis por ativar os métodos que residem nos objetos, pois são eles – os métodos – que definem como um determinado objeto deve reagir às mensagens a ele enviadas, representando o seu comportamento (DEITEL e DEITEL, 2017).

Premissa 5

"Os objetos de um sistema desenvolvido em POO precisam ser criados (instanciados)."

É importante observar que a criação de objetos durante a execução do programa requer o comando de criação, tal comando segue a mesma sintaxe da utilização de variáveis. Todavia, vale ressaltar que a criação de um objeto não se trata apenas de uma declaração de variável, pois o processo necessita de comando específico o qual se denomina **instanciação** (SANTOS 2003).

Objetos são criados por meio de instanciação que utiliza o **construtor** para definir o estado inicial dos objetos.

Premissa 6

"O encapsulamento permite a ocultação da complexidade da implementação ao mesmo tempo que protege as propriedades dos objetos."

O encapsulamento é a técnica utilizada para restringir o acesso aos atributos, métodos ou até mesmo à própria classe. Nesse caso, os detalhes da implementação ficam ocultos ao usuário da classe, dessa forma, o usuário passa a utilizar os métodos de uma classe sem se preocupar com detalhes sobre como o método foi implementado internamente (CARVALHO e TEIXEIRA, 2012).

Ocultar aspectos que não precisam ser mostrados ao usuário é caracterizado como um grande trunfo da POO em relação a outros paradigmas de programação, pois dessa maneira, a complexidade do código permanece transparente para o usuário, de forma que apenas reutiliza a interface previamente implementada.

O encapsulamento de atributos, métodos ou classes se dá por meio da utilização de padrões de acesso, os quais são definidos no memento em que tais elementos são implementados. Em POO podem ser utilizados três formas de padrão de acesso: público; privado e protegido.

Contudo, o acesso e utilização dos atributos encapsulados de uma classe é possível por meio dos métodos acessores. Tais métodos têm a função de servir de interface de acesso ao conteúdo existente nos atributos da classe. Nesse caso, atributos encapsulados devem ter um método que obtenha o seu valor atual (método *get*) e um método que possibilite alterar o valor do atributo (método *set*) (CARVALHO e TEIXEIRA, 2012).

Premissa 7

"Herança é um ponto fundamental do paradigma."

Não é ao acaso que a característica de herança seja um dos principais conceitos do paradigma orientado a objetos, pois, por intermédio dela é possível a reutilização de código anteriormente desenvolvido. Essa característica agiliza o processo de desenvolvimento e manutenção de produtos de software em linguagem orientada a objetos. Todavia, é importante ressaltar que a herança não se restringe à possibilidade de reutilização de códigos, pois também, proporciona ao projetista e ao desenvolvedor maior clareza e organização durante todas as fases do processo de criação do software.

Assim, a herança supõe a existência de uma classe principal (superclasse) e uma hierarquia abaixo dela formada por classes derivadas. As classes derivadas (subclasses) herdam o código da superclasse acrescentando-lhe

características próprias na forma de novos atributos, de operações (métodos) ou pelo aperfeiçoamento/adequação de operações.

Premissa 8

"Um objeto pode ter várias formas (Polimorfismo) pela reimplementação ou sobrecarga de métodos, em uma relação de herança."

Polimorfismo *significa "várias formas" e é uma característica fundamental da POO, pois é a capacidade de modificar comportamentos de métodos dos objetos. Segundo (Tucker e Noonam, 2010, p. 323) "em linguagens orientadas a objetos, polimorfismo refere-se à ligação tardia de uma chamada a uma ou várias diferentes implementações de um método em uma hierarquia de herança". Assim, o conceito de polimorfismo está relacionado com a possibilidade da relação de herança entre as classes, pois, a mudança de comportamentos de objetos pertencentes a uma mesma estrutura hierárquica e, dispostos em diferentes níveis, é o que possibilita o polimorfismo.*

5.2) Atividade Proposta

Passo 1: Utilizando o Greenfoot, vamos construir na íntegra o projeto "Colheita das Maçãs" conforme Descrição, Figura 1 e Vídeo 1 abaixo.

Descrição do Jogo "Colheita das Maçãs"

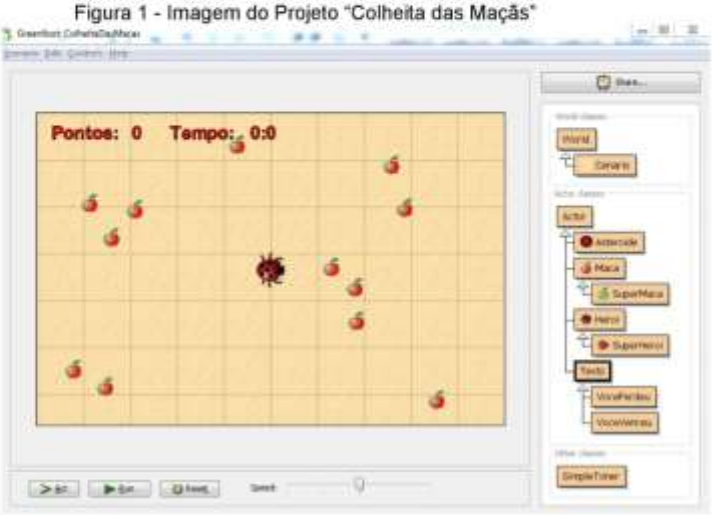
O jogo ocorre em um cenário 2D, em que maçãs são distribuídas aleatoriamente e precisam ser colhidas pelo personagem principal (Herói) representado pela imagem de uma joaninha. A cada maçã colhida a pontuação do herói é incrementada em uma unidade.

Como desafio, o personagem principal precisa desviar-se de uma chuva de meteoros que caem aleatoriamente no cenário, pois caso algum se choque com o herói o jogo acaba e ele perde.

Para incentivo, ocorre também de maneira aleatória uma chuva de SuperMaçãs representadas pela imagem de uma maçã verde. Caso o herói consiga colher uma dessas supermaçãs, passa para o status de SuperHerói (mudando de imagem e comportamento) e a partir daí a cada maçã colhida, sua pontuação é incrementada

em duas unidades. O herói é vitorioso quando consegue computar um total de 10 pontos.

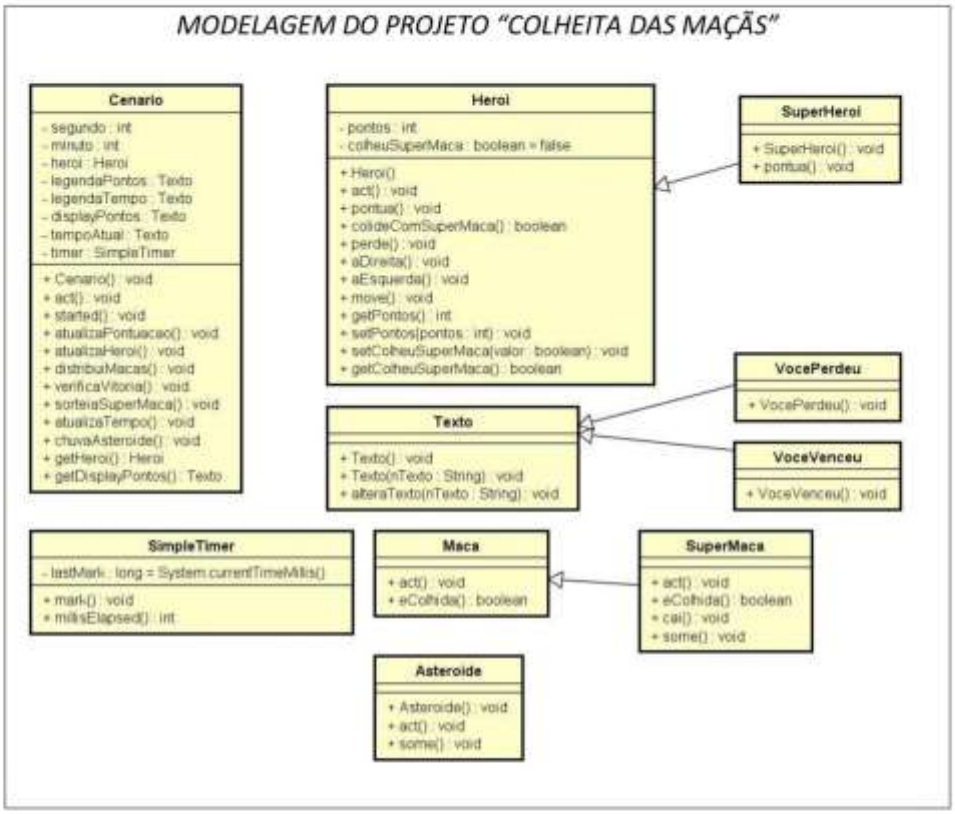
Figura 1 - Imagem do Projeto "Colheita das Maçãs"



Vídeo 1 - Vídeo Projeto

Fonte: https://youtu.be/cVZZheoeM_E

Passo 2: Realize a implementação conforme modelagem UML e descrição abaixo.



5.3) Descrição das classes

I. Descrição da classe "Cenario"

Cenario

É a classe principal do jogo – *extends* de World, nela todos os objetos visuais necessários à proposta do jogo precisam ser instanciados e adicionados em suas respectivas posições. Possui a dimensão de 600X400 px e uma imagem de fundo à escolha do programador.

a. **Construtor (Cenario()):** No construtor da classe cenário, os objetos visuais devem ser instanciados e adicionados em suas respectivas posições utilizando o método *addObject(Actor object, int x, int y)* que é herdado da classe *World*. Ainda, deve ocorrer a chamada ao método *distribuiMacas()* para inserir as maçãs no cenário do jogo.

b. **act():** Esse método é responsável por executar as ações da classe em loop infinito enquanto o jogo estiver rodando, portando todas as ações necessárias à execução do jogo devem ser chamadas nele: atualização do tempo (minutos e segundos), *atualizaTempo(), atualizaHeroi(), verificaVitória(), chuvaAsteroide(), sorteiaSuperMaca(), atualizaPontuacao()*.
c. **started():** esse método é herdado da classe *Greenfoot* e para esse jogo, tem a função de inicializar a contagem de tempo, pela chamada do método *mark()* para o objeto *timer [timer.mark()]*. O método *started()* é chamado uma única vez ao início da execução de um jogo no ambiente *Greenfoot* – nesse caso, serve de "gatilho" para iniciar a contagem de tempo.
d. **atualizaPontuacao():** Tem a função de atualizar a pontuação visível no cenário – nesse caso utiliza o objeto *displayPontos*, invocando o método *alteraTexto(String nTexto)*, passando por parâmetro os pontos do objeto *herói [this.getHeroi().getPontos()]*.
e. **atualizaHeroi():** método que tem a função de verificar se o objeto *heroi* se chocou com um objeto do tipo superMaca [*heroi.getColheuSuperMaca()*] – caso isso ocorra o próprio objeto é promovido a SuperHeroi. Para que isso funcione adequadamente no ambiente Greenfoot, primeiro é necessário que o estado do objeto seja salvo para que as informações não se percam (coordenadas x e y, pontos e rotação), na sequência o objeto deve ser removido do cenário [*removeObject(heroi)]* e reinstanciado na forma de um SuperHeroi [*heroi = new SuperHeroi()]* e o estado salvo deve ser aplicado ao objeto *heroi* inserindo-o novamente no cenário [*addObject(heroi, x, y)]* e setando os valores salvos de *pontos* e *rotação*.
f. **distribuiMacas():** método responsável por distribuir aleatoriamente doze maçãs no cenário. Para isso, utiliza um laço de repetição que sorteia coordenadas x e y utilizando o método *Greenfoot.getRandomNumber(int alcance)* ou *Math.random()* e adicionando objetos do tipo Maca no cenário [*addObject(new Maca(),x, y)*].
g. **verificaVitoria():** verifica se o atributo pontos do objeto heroi é maior ou igual a 10 – caso a expressão seja verdadeira um objeto do tipo VoceVenceu é instanciado na coordenada x=300 e y=200 do cenário e o jogo é finalizado [*Greenfoot.stop()*].
h. **sorteiaSuperMaca():** O objetivo desse método é sortear SuperMaca's no cenário – enquanto o objeto herói ainda pertencer a classe Heroi [*heroi.getClass().getName().equals("Heroi")*], pois quando ele já for um SuperHeroi, não há necessidade de continuar sorteando. Para fazer isso é necessário sortear a coordenada x para inserir a supermaçã visto que a coordenada y pode ser sempre igual a 0 (zero). Nesse caso, utiliza-se o comando *int x = Greenfoot.getRandomNumber(int alcance);* sendo que o valor máximo para x deve ser 600 que é a extensão do 'eixo x' do cenário. **Dica**: se atribuir o alcance de apenas 600 para o método getRandomNumber, muitos objetos serão inseridos, inviabilizando a execução, nesse caso deve-se atribuir

um valor significativamente maior (ex: *30000*) e selecionar (*if*) apenas aqueles que forem menores que 600 – outra coisa que pode-se fazer para diminuir o numero de itens sorteados é selecionar apenas aqueles que forem divisíveis por determinado número (ex: *x%6==0*).

i. **atualizaTempo():** apenas atualiza os atributos segundo e minuto, capturando o tempo do método millisElapsed() do objeto timer.
j. **chuvaAsteroide():** faz "chover" asteroides no cenário, para isso é necessário sortear a coordenada x para inserir um asteroide visto que a coordenada y pode ser sempre igual a 0 (zero). Nesse caso, utiliza-se o comando *int x = Greenfoot.getRandomNumber(int alcance);* sendo que o valor máximo para x deve ser 600 que é a extensão do 'eixo x' do cenário. **Dica**: se atribuir o alcance de apenas 600 para o método getRandomNumber, muitos objetos serão inseridos, inviabilizando a execução, nesse caso pode-se atribuir um valor significativamente maior (ex: *10000*) e selecionar (*if*) apenas aqueles que forem menores que 600 – outra coisa que deve-se fazer para diminuir o numero de itens sorteados é selecionar apenas aqueles que forem divisíveis por determinado número (ex: *x%6==0*).
k. **getHeroi():** retorna o atributo heroi.
l. **getDisplayPontos():** retorna o atributo displayPontos.

II. Descrição da Classe Heroi

Extends de Actor - Possui dois atributos: pontos - para armazenar a pontuação do objeto e colheuSuperMaca para indicar se o objeto chocou-se com um objeto do tipo SuperMaca (valor *default* é *false*).

a. **Construtor (Heroi()):** Possui apenas o comando de atribuição da imagem *ladybug_02.png* para o objeto [*setImage(String imagem)*];
b. **act():** Esse método é responsável por executar as ações dos objetos da classe em loop infinito enquanto o jogo estiver rodando, portando todas as ações que devam ser executadas ou verificadas de maneira contínua devem ser chamadas dentro desse método: *move(), aDireita(), aEsquerda(), pontua(), perde(), colideComSuperMaca()*.
c. **pontua():** método responsável por verificar se o personagem colidir com um objeto do tipo Maca [*getOneIntersectingObject(Maca.class)*], uma unidade deve ser incrementada ao seu atributo pontos e o som "*slurp.wav*" deve ser executado [*Greenfoot.playSound(String)*];
d. **colideComSuperMaca**(): caso o objeto colida com um objeto do tipo SuperMaca [*getOneIntersectingObject(SuperMaca.class)*], seu atributo colheuSuperMaca deve receber o valor *true* e o som "*notify.wav*" deve ser executado [*Greenfoot.playSound(String)*] – o método retorna o valor do atributo colheuSuperMaca;

e. **perde():** verifica se o objeto colidir com uma instância da classe Asteroide, um objeto da classe VocePerdeu é instanciado na coordenada *x=300* e *y=200* e o jogo é encerrado [*Greenfoot.stop()*];
f. **aDireita():** tem a função de rotacionar [*setRotation(int)*] o objeto três unidades à direita caso a seta à direita (*right*) for pressionada [*isKeyDown(String key)*].
g. **aEsquerda():** tem a função de rotacionar [*setRotation(int)*] o objeto três unidades à esquerda caso a seta à esquerda (*left*) for pressionada [*isKeyDown(String key)*].
h. **move():** tem por objetivo dar movimento ao objeto. Por padrão está sempre se movimentando à frente [*move(2)*] – sua direção no cenário é modificada pela sua rotação[*aEsquerda()* e *aDireita()*]. Caso a tecla seta para baixo (*down*) seja pressionada o objeto anda para trás, caso a tecla espaço (*space*) for pressionada o objeto para.
i. **getPontos():** retorna o atributo pontos;
j. **setPontos(int):** altera o valor do atributo pontos;
k. **setColheuSuperMaca (boolean):** altera o valor do atributo colheuSuperMaca;
l. **getColheuSuperMaca ():** altera o valor do atributo colheuSuperMaca;

III. Descrição da classe SuperHeroi

Extends de Heroi - é uma especialização da classe Heroi, contendo métodos melhorados.

a. **Construtor (SuperHeroi()):** Possui o comando de atribuição da imagem *ladybug1.png* para o objeto [*setImage(String imagem)*] e o valor padrão para o atributo colheuSuperMaca é *true;*
b. **pontua():** método responsável por verificar se o personagem colidir com um objeto do tipo Maca[*getOneIntersectingObject(Maca.class)*], duas unidades devem ser incrementadas ao seu atributo pontos e o som "*slurp.wav*" deve ser executado[*Greenfoot.playSound(String)*];

IV. Descrição da classe Texto

Extends de Actor – não possui atributos. Tem a finalidade de criar objetos que apresentam informações textuais no cenário.

a. **Construtor (Texto()):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "Texto:", com tamanho 30px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

b. **Construtor (Texto(String)):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é passado por parâmetro com tamanho 30px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].
c. alteraTexto(String): método que tema função de alterar o texto exibido pelo objeto, sendo que o novo texto é passado por parâmetro. Todavia é necessário criar um objeto do tipo GreenfootImage a partir do texto informado com tamanho 30px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

V. Descrição da classe VocePedeu

Extends de Texto – não possui atributos. Tem a finalidade de criar objeto contendo o texto que indica derrota do personagem principal.

a. **Construtor (VocePerdeu()):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "Você Perdeu", com tamanho 30px, na cor vermelha, fundo amarelo e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

VI. Descrição da classe VoceVenceu

Extends de Texto – não possui atributos. Tem a finalidade de criar objeto contendo o texto que indica vitória do personagem principal.

a. **Construtor (VoceVenceu()):** Instancia um objeto do tipo GreenfootImage cujo texto padrão é "Você Venceu", com tamanho 30px, na cor vermelha, fundo transparente e contorno das letras na cor preta. O objeto instanciado é utilizado na forma de imagem [*setImage(GreenfootImage)*].

VII. Descrição da classe Maca

Extends de Actor – não possui atributos. Tem a finalidade de criar objeto do tipo maçã para serem colhidos pelo personagem principal.

a. **act():** Esse método é responsável por executar as ações dos objetos da classe em loop infinito enquanto o jogo estiver rodando. Nesse caso, nesse método ocorre apenas a chamada do método *eColhida()*;

b. **eColhida():** tem a finalidade de verificar se o objeto colide com uma instância do tipo Heroi [*isTouching(Heroi.class)*] e, caso isso ocorra, ele é removido do cenário [*getWorld().removeObject(this)*] e retorna o valor *true*, caso contrário retorna o valor *false*.

VIII. Descrição da classe SuperMaca

Extends de Maca – não possui atributos. Tem a finalidade de criar objeto do tipo maçã para serem colhidos pelo personagem principal.

a. **act():** esse método sofre reimplementação em relação ao mesmo método da sua classe ancestral, pois além de executar o mesmo procedimento descrito no método act() da classe Maca [*super.act()*], os objetos criados a partir dessa classe devem estar caindo no cenário 2 px por ciclo, nesse caso deve ocorrer a chamada do método *cai()* e *some()*.
b. **eColhida():** esse método sofre reimplementação em relação ao mesmo método da sua classe ancestral, pois além de executar o mesmo procedimento descrito no método da superclasse, tem a finalidade de, ao verificar se o objeto colide com uma instância do tipo Heroi [*isTouching(Heroi.class)*], removê-lo do cenário [*getWorld().removeObject(this)*] e alterar o atributo colheuSupermaca do objeto do tipo Heroi que o tocou para o valor *true* *[((Cenario)getWorld()).getHeroi().setColheuSuperMaca(true)]* - retorna o valor *true*, caso contrário retorna o valor *false*.
c. **cai():** tem a finalidade de movimentar continuamente o objeto 2px por ciclo no eixo y – usa-se o método setLocation(x,y) [*this.setLocation(this.getX(), this.getY()+2)];*
d. **some():** verifica se o objeto chegou ao final do cenário [*isAtEdge()*] e, caso afirmativo, remove-o do "*mundo*" [*getWorld.removeObject(this)*];

IX. Descrição da classe Asteroide

Extends de Actor – não possui atributos. Tem a finalidade de criar objeto do tipo Asteroide que serão sorteados em posições aleatórias no cenário.

a. **Construtor (Asteroide()):** Possui o comando de rotação [*turn(90)*] que tem a finalidade de girar o objeto em 90º para que deslize de cima para baixo o cenário;

b. **act():** nesse método ocorre a chamada de dois métodos apenas, *move(int)* que é um método herdado de Actor e tem a finalidade de movimentar continuamente o objeto 2px por ciclo [*move(2)*];
c. **some():** verifica se o objeto chegou ao final do cenário [*isAtEdge()*] e, caso afirmativo, remove-o do "*mundo*" [*getWorld.removeObject(this)*];

X. Descrição da classe SimpleTimer

Essa classe é nativa do Greenfoot, precisa ser importada para dentro do projeto – isso é possível pelo menu Edit >Import class... Tem a finalidade de contabilizar o tempo decorrido em milissegundos. Não é necessário alterá-la, pois já tem o atributo lastMark, que armazena o tempo decorrido e os métodos *mark()* e *millisElapsed()*.

a. **mark():** reinicia o valor da variável lastMark;
b. **millisElapsed ():** retorna o tempo decorrido que está armazenado no atributo lastMark;

As imagens e sons necessários para construir o projeto podem ser baixados do link: https://drive.google.com/drive/folders/1_He062HnfW2Csv0-cBvkXm6UQhG-6nlg?usp=sharing

Desafio

A partir do que aprendemos até aqui sobre orientação a objetos e sobre Greenfoot, implemente funcionalidades extras ao projeto, adicionando recursos na classe SuperHeroi, modificando o funcionamento dos métodos existentes ou inserindo novos.

Sugestão de funcionalidades aprimoradas:

- Movimento mais rápido;
- Proteção contra asteroide;
- "teletransporte" – mudar de lugar a fim de fugir de asteroides.

6 Ferramentas

- Software Greenfoot;

7 Avaliação

1) A partir de tudo que foi estudado sobre o POO e da implementação do projeto "Colheita das Maçãs", procure identificar, ilustrar e exemplificar onde podemos perceber a consolidação de cada umas das premissas (de 1 a 8) descritas no inicio dessa unidade.

Referências

CARDOSO, Caique. **Orientação a objetos na prática:** aprendendo orientação a objetos com Java. Rio de Janeiro: Ciência Moderna, 2006

CARVALHO, V. A. de. TEIXEIRA, G. F. **Programação orientada a objetos**: Curso técnico de informática. Colatina: IFES, 2012.

DEITEL, P.; DEITEL, H. **Java**: como programar. São Paulo: Pearson Education do Brasil, 2017.

KEOGH, James Edward; GIANNINI, Mario. **OOP desmitificado:** programação orientada a objetos. Rio de Janeiro: Alta Books, 2005.

OMG. **About OMG**. 2017. Disponível em: < http://www.omg.org/about/index.htm>. Acessado em: 06 nov 2017.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Elsevier, 2003.

TUCKER, Allen B.; NOONAN, Robert E. **Linguagens de programação**: princípios e paradigmas. 2.ed. Porto Alegre : AMGH, 2010.

APÊNDICE G – QUESTIONÁRIO DE OPINIÃO SOBRE O USO DO GREENFOOT COMO FERRAMENTA AUXILIAR NO ENSINO DE POO

ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS NA EDUCAÇÃO PROFISSIONAL POR MEIO DO DESENVOLVIMENTO DE JOGOS APOIADO PELO AMBIENTE GREENFOOT

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO DE USO DE IMAGEM E DEPOIMENTO

Prezado (a) Senhor (a)

Você está sendo convidado(a) a participar de uma pesquisa sobre as possibilidades de uso do desenvolvimento de jogos para a aprendizagem dos conceitos de Programação Orientada a Objetos apoiado pelo uso do ambiente Greenfoot, que está sendo desenvolvida por Cleitom José Richter, do Programa de Pós-Graduação em Tecnologias Educacionais e Rede (PPGTER), da Universidade Federal de Santa Maria (UFSM), intitulado "ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS NA EDUCAÇÃO PROFISSIONAL POR MEIO DO DESENVOLVIMENTO DE JOGOS APOIADO PELO AMBIENTE GREENFOOT".

O presente questionário tem como objetivo coletar a opinião dos estudantes quanto a utilização do ambiente de programação Greenfoot e do desenvolvimento de jogos na aprendizagem dos conceitos de POO. Essa ação é parte integrante da referida pesquisa de Mestrado que tem como propósito compreender como a utilização do desenvolvimento de jogos pode contribuir na apropriação dos conceitos de POO.

Para o estudo foram selecionadas duas turmas do curso Técnico em Informática integrado ao Ensino Médio(3º ano)do IFFar campus Santo Augusto, sendo que cada turma é composta de aproximadamente 28 estudantes, com atividades na disciplina de Programação II. Em ambas as turmas, as disciplinas citadas são ministradas pelo professor pesquisador.

Para responder a esse questionário você precisa ser estudante de uma das turmas selecionadas e ter participados das dinâmicas de aula referentes ao estudo de POO a partir do desenvolvimento jogos com Greenfoot.

A partir desse trabalho espera-se melhorar/facilitar significativamente o entendimento dos alunos acerca dos conceitos de orientação a objetos, bem como, promover melhoria nas intervenções didáticas no ensino de programação orientada a objetos. A presente pesquisa é conduzida pelo Mestrando Cleitom José Richter (cleitom.richter@iffarroupilha.edu.br) sob orientação da Professora Doutora Giliane Bernardi (giliane@inf.ufsm.br).

Será mantido total sigilo sobre sua identificação e participação, tanto na apresentação dos resultados como tabulação dos dados.

Sua participação é livre e voluntária, nesse caso, você não terá nenhum gasto e também não receberá nenhum pagamento por participar desse estudo. Desse modo, deixamos claro que poderás interromper o preenchimento do formulário online a qualquer momento, retirando esse consentimento sem penalidade alguma. Você não terá benefício direto, mas o estudo poderá promover melhoria em futuras intervenções didáticas no ensino de programação orientada a objetos.

*Obrigatório

1. **Concordas em participar da pesquisa?** *
   Marcar apenas uma oval.

   ◯ Sim
   ◯ Não *Pare de preencher este formulário.*

ENSINO DE PROGRAMAÇÃO ORIENTADA A OBJETOS NA EDUCAÇÃO PROFISSIONAL POR MEIO DO

DESENVOLVIMENTO DE JOGOS APOIADO PELO AMBIENTE GREENFOOT

2. **Endereço de e-mail** *

3. **Ao estudar conceitos de POO, prefere que o professor utilize uma metodologia tradicional, com o uso de apostila, slides e exercícios?** *
Marcar apenas uma oval.

- a) Sim, me sinto mais à vontade
- b) Talvez, prefiro que faça a união metodologias tradicionais e intervenções mais dinâmicas
- c) Não, prefiro que sempre utilize metodologias dinâmicas com trabalhos em grupos, debates e que o estudante tenha autonomia em aprender ao seu tempo

4. **Sobre a ferramenta Greenfoot, você conseguiu perceber a utilização dos principais conceitos de orientação à objetos (classe, objeto, instanciação, construtor, encapsulamento, herança, mensagem e polimorfismo)?** *
Marcar apenas uma oval.

- a) Sim, todos os conceitos
- b) Muitos desse s conceitos
- c) Poucos desses conceitos
- d) Nenhum desses conceitos

5. **Na sua opinião, a utilização da ferramenta Greenfoot facilitou o entendimento dos conceitos de POO?** *
Marcar apenas uma oval.

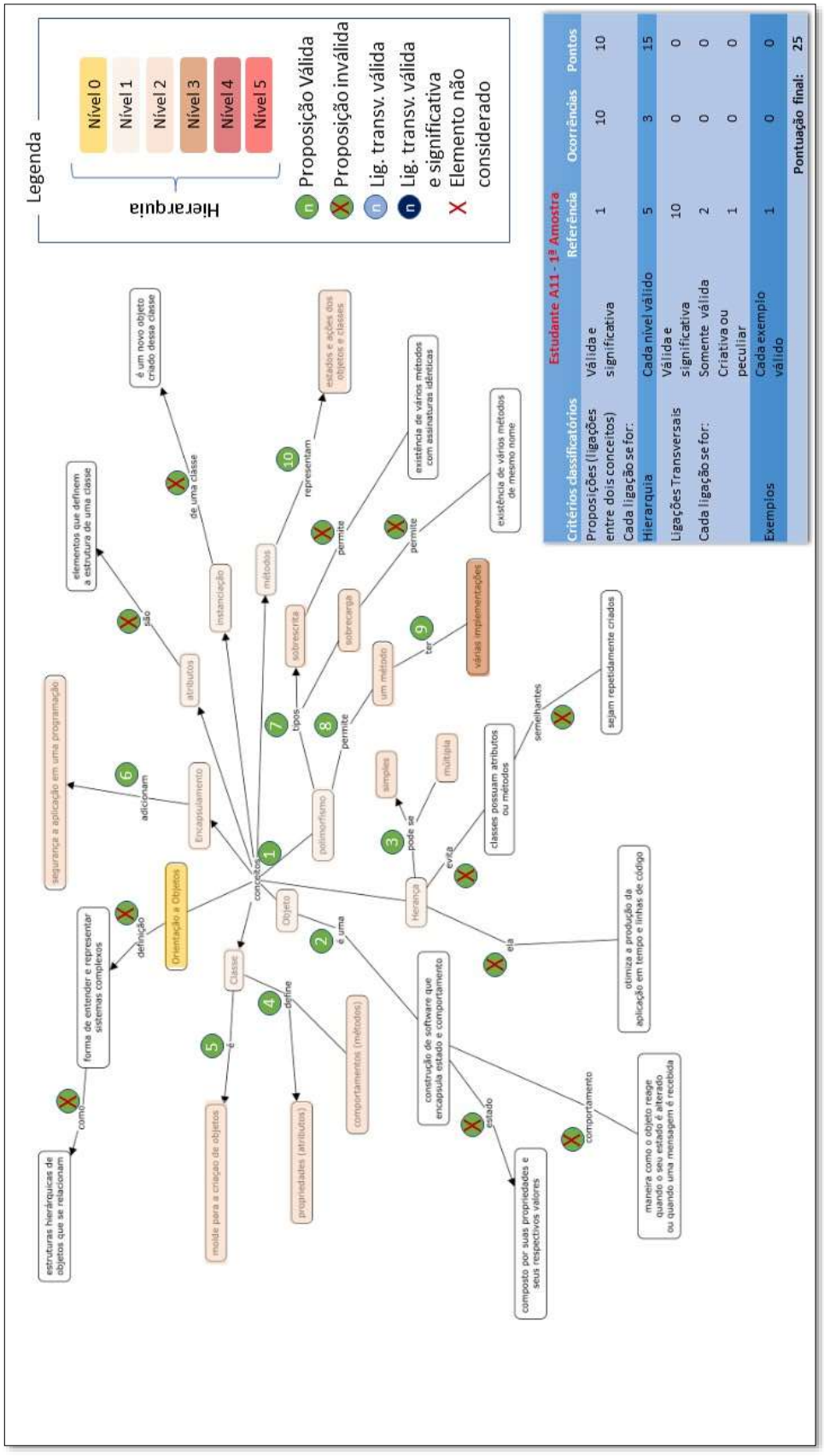
- a) Sim, plenamente
- b) Sim, razoavelmente
- c) Indiferente
- d) Não, senti muitas dificuldades

6. **Se você fosse atribuir uma nota para as atividades que envolveram o ensino de POO com o uso do Greenfoot, que nota daria?** *
Marcar apenas uma oval.

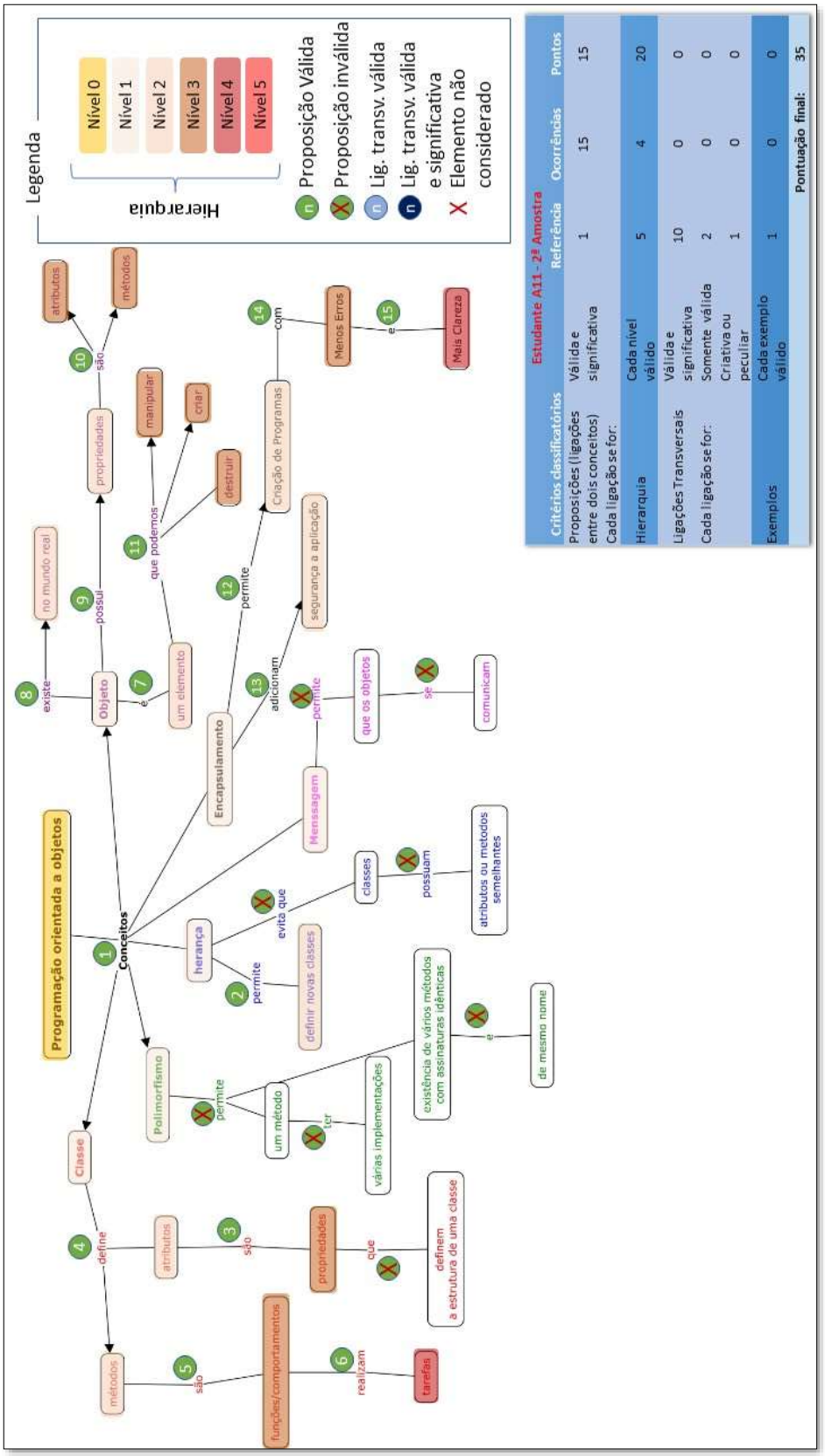
	1	2	3	4	
Regular	◯	◯	◯	◯	Excelente

7. **Qual é a sua opinião sobre as atividades desenvolvidas (críticas ou sugestões):**

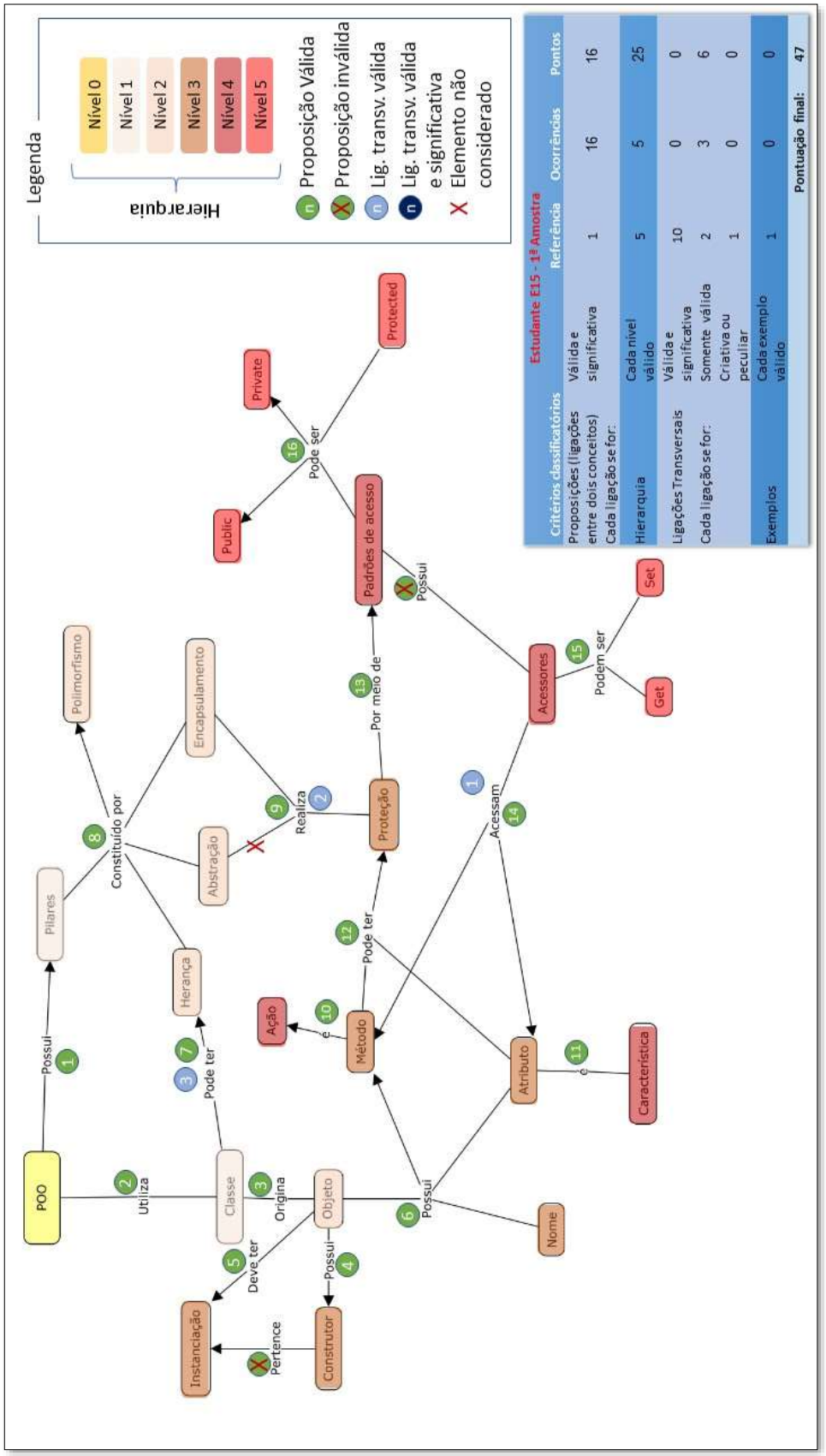
APÊNDICE H – AVALIAÇÃO MAPA CONCEITUAL ESTUDANTE A11 – 1ª AMOSTRA



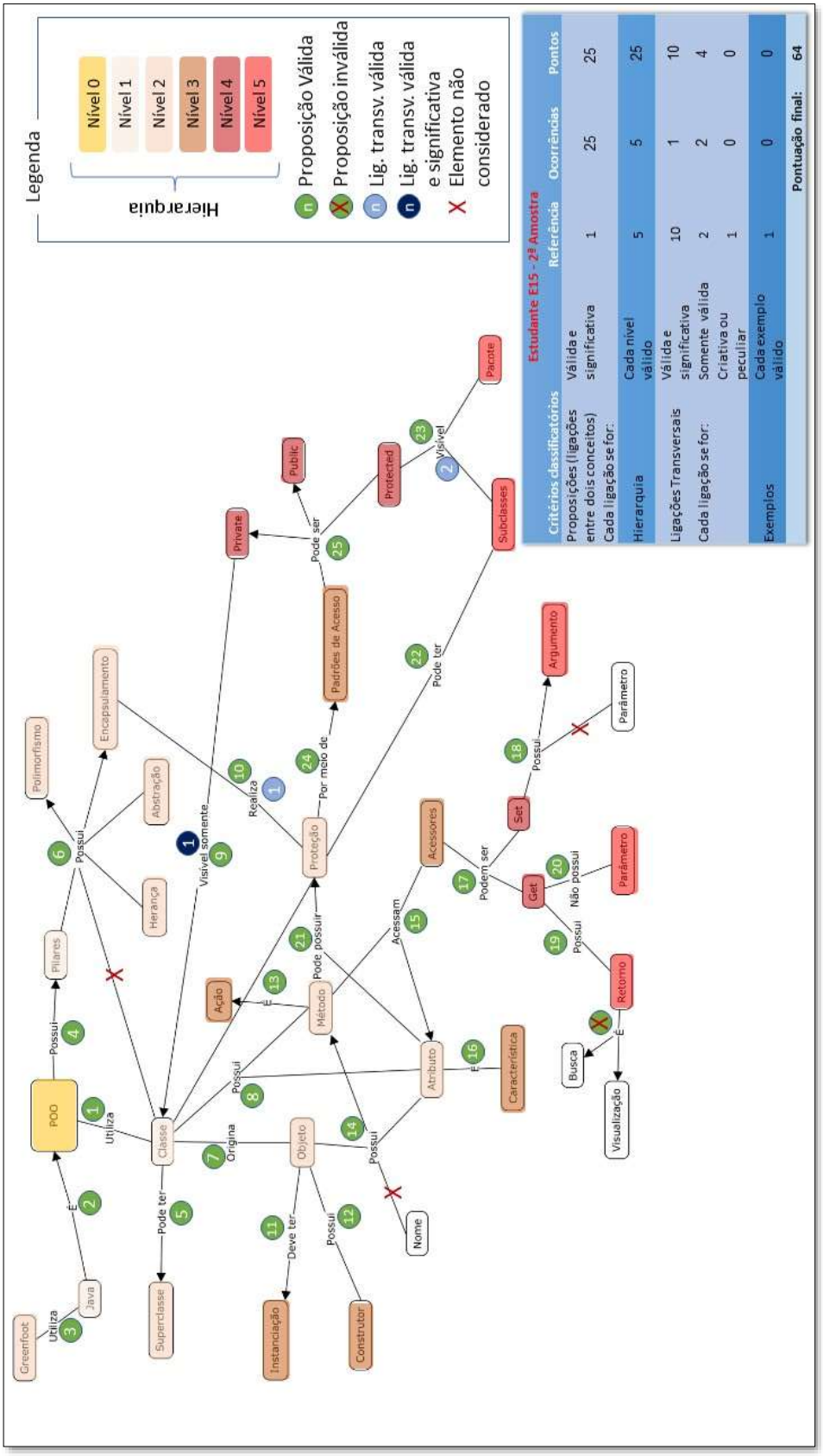
APÊNDICE I – AVALIAÇÃO MAPA CONCEITUAL ESTUDANTE A11 – 2ª AMOSTRA



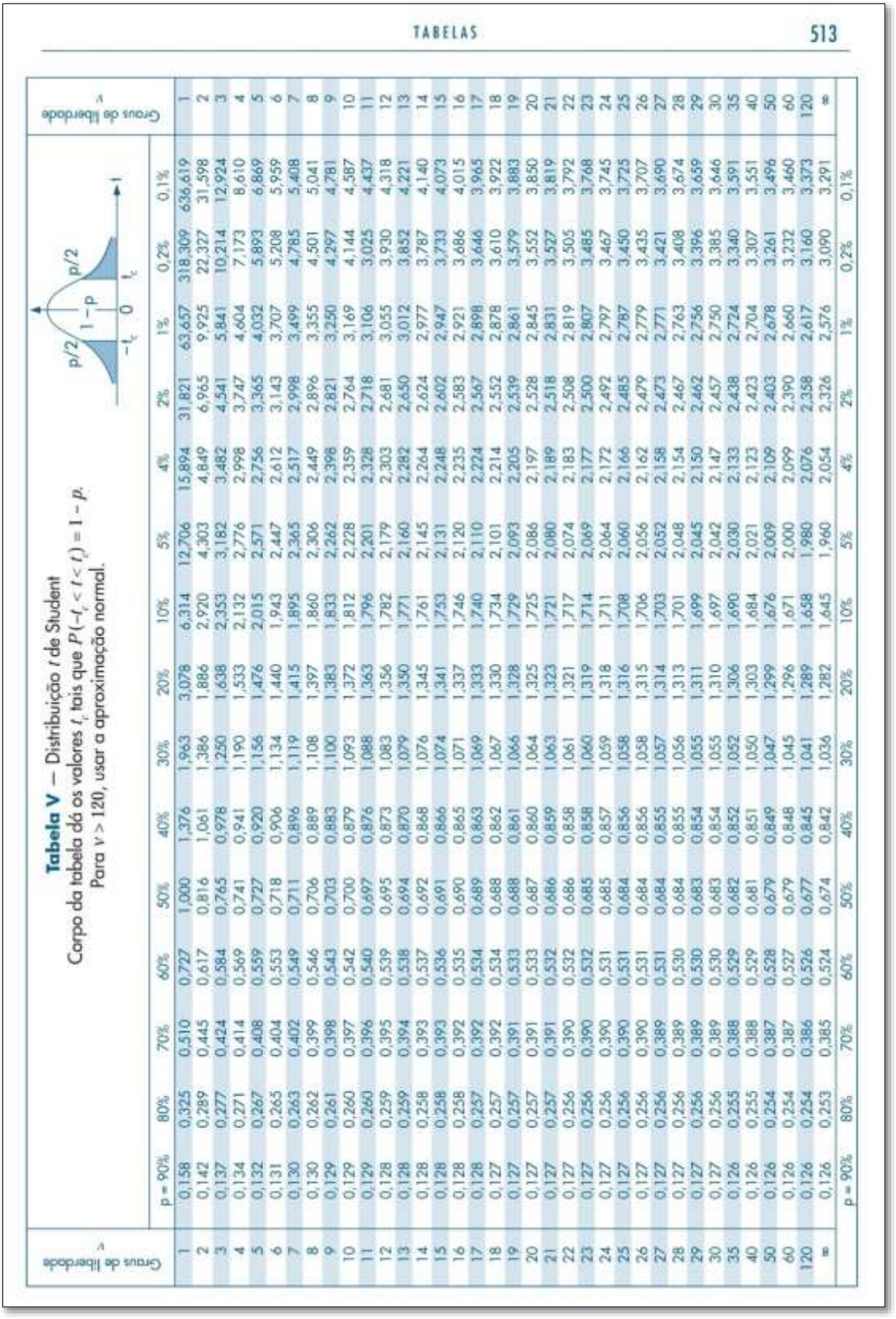
APÊNDICE J – AVALIAÇÃO MAPA CONCEITUAL ESTUDANTE E15 – 1ª AMOSTRA



APÊNDICE K – AVALIAÇÃO MAPA CONCEITUAL ESTUDANTE E15 – 2ª AMOSTRA



ANEXO A – TABELA VALORES CRÍTICOS DA DISTRIBUIÇÃO T



TABELAS 513

Tabela V — Distribuição *t* de Student

Corpo da tabela dá os valores t_c tais que $P(-t_c < t < t_c) = 1 - p$.
Para $\nu > 120$, usar a aproximação normal.

Graus de liberdade ν	p = 90%	80%	70%	60%	50%	40%	30%	20%	10%	5%	4%	2%	1%	0,2%	0,1%	Graus de liberdade ν
1	0,158	0,325	0,510	0,727	1,000	1,376	1,963	3,078	6,314	12,706	15,894	31,821	63,657	318,309	636,619	1
2	0,142	0,289	0,445	0,617	0,816	1,061	1,386	1,886	2,920	4,303	4,849	6,965	9,925	22,327	31,598	2
3	0,137	0,277	0,424	0,584	0,765	0,978	1,250	1,638	2,353	3,182	3,482	4,541	5,841	10,214	12,924	3
4	0,134	0,271	0,414	0,569	0,741	0,941	1,190	1,533	2,132	2,776	2,998	3,747	4,604	7,173	8,610	4
5	0,132	0,267	0,408	0,559	0,727	0,920	1,156	1,476	2,015	2,571	2,756	3,365	4,032	5,893	6,869	5
6	0,131	0,265	0,404	0,553	0,718	0,906	1,134	1,440	1,943	2,447	2,612	3,143	3,707	5,208	5,959	6
7	0,130	0,263	0,402	0,549	0,711	0,896	1,119	1,415	1,895	2,365	2,517	2,998	3,499	4,785	5,408	7
8	0,130	0,262	0,399	0,546	0,706	0,889	1,108	1,397	1,860	2,306	2,449	2,896	3,355	4,501	5,041	8
9	0,129	0,261	0,398	0,543	0,703	0,883	1,100	1,383	1,833	2,262	2,398	2,821	3,250	4,297	4,781	9
10	0,129	0,260	0,397	0,542	0,700	0,879	1,093	1,372	1,812	2,228	2,359	2,764	3,169	4,144	4,587	10
11	0,129	0,260	0,396	0,540	0,697	0,876	1,088	1,363	1,796	2,201	2,328	2,718	3,106	3,025	4,437	11
12	0,128	0,259	0,395	0,539	0,695	0,873	1,083	1,356	1,782	2,179	2,303	2,681	3,055	3,930	4,318	12
13	0,128	0,259	0,394	0,538	0,694	0,870	1,079	1,350	1,771	2,160	2,282	2,650	3,012	3,852	4,221	13
14	0,128	0,258	0,393	0,537	0,692	0,868	1,076	1,345	1,761	2,145	2,264	2,624	2,977	3,787	4,140	14
15	0,128	0,258	0,393	0,536	0,691	0,866	1,074	1,341	1,753	2,131	2,248	2,602	2,947	3,733	4,073	15
16	0,128	0,258	0,392	0,535	0,690	0,865	1,071	1,337	1,746	2,120	2,235	2,583	2,921	3,686	4,015	16
17	0,128	0,257	0,392	0,534	0,689	0,863	1,069	1,333	1,740	2,110	2,224	2,567	2,898	3,646	3,965	17
18	0,127	0,257	0,392	0,534	0,688	0,862	1,067	1,330	1,734	2,101	2,214	2,552	2,878	3,610	3,922	18
19	0,127	0,257	0,391	0,533	0,688	0,861	1,066	1,328	1,729	2,093	2,205	2,539	2,861	3,579	3,883	19
20	0,127	0,257	0,391	0,533	0,687	0,860	1,064	1,325	1,725	2,086	2,197	2,528	2,845	3,552	3,850	20
21	0,127	0,257	0,391	0,532	0,686	0,859	1,063	1,323	1,721	2,080	2,189	2,518	2,831	3,527	3,819	21
22	0,127	0,256	0,390	0,532	0,686	0,858	1,061	1,321	1,717	2,074	2,183	2,508	2,819	3,505	3,792	22
23	0,127	0,256	0,390	0,532	0,685	0,858	1,060	1,319	1,714	2,069	2,177	2,500	2,807	3,485	3,768	23
24	0,127	0,256	0,390	0,531	0,685	0,857	1,059	1,318	1,711	2,064	2,172	2,492	2,797	3,467	3,745	24
25	0,127	0,256	0,390	0,531	0,684	0,856	1,058	1,316	1,708	2,060	2,166	2,485	2,787	3,450	3,725	25
26	0,127	0,256	0,390	0,531	0,684	0,856	1,058	1,315	1,706	2,056	2,162	2,479	2,779	3,435	3,707	26
27	0,127	0,256	0,389	0,531	0,684	0,855	1,057	1,314	1,703	2,052	2,158	2,473	2,771	3,421	3,690	27
28	0,127	0,256	0,389	0,530	0,684	0,855	1,056	1,313	1,701	2,048	2,154	2,467	2,763	3,408	3,674	28
29	0,127	0,256	0,389	0,530	0,683	0,854	1,055	1,311	1,699	2,045	2,150	2,462	2,756	3,396	3,659	29
30	0,127	0,256	0,389	0,530	0,683	0,854	1,055	1,310	1,697	2,042	2,147	2,457	2,750	3,385	3,646	30
35	0,126	0,255	0,388	0,529	0,682	0,852	1,052	1,306	1,690	2,030	2,133	2,438	2,724	3,340	3,591	35
40	0,126	0,255	0,388	0,529	0,681	0,851	1,050	1,303	1,684	2,021	2,123	2,423	2,704	3,307	3,551	40
50	0,126	0,254	0,387	0,528	0,679	0,849	1,047	1,299	1,676	2,009	2,109	2,403	2,678	3,261	3,496	50
60	0,126	0,254	0,387	0,527	0,679	0,848	1,045	1,296	1,671	2,000	2,099	2,390	2,660	3,232	3,460	60
120	0,126	0,254	0,386	0,526	0,677	0,845	1,041	1,289	1,658	1,980	2,076	2,358	2,617	3,160	3,373	120
∞	0,126	0,253	0,385	0,524	0,674	0,842	1,036	1,282	1,645	1,960	2,054	2,326	2,576	3,090	3,291	∞
	p = 90%	80%	70%	60%	50%	40%	30%	20%	10%	5%	4%	2%	1%	0,2%	0,1%	