Relatórios Técnicos do Programa de Pós-Graduação em
Tecnologias Educacionais em Rede
Universidade Federal de Santa Maria

ISSN 2675-0309

PPGTER/DES.02.2019.UES

Aprendendo orientação à Objetos com Greenfoot: Unidades de Estudo

Autores
Cleitom José Richter
cleitom.richter@iffarroupilha.edu.br

Giliane Bernardi
giliane@inf.ufsm.br

Versão 1.0
Status: Final
Distribuição: Externa
MARÇO 2019

Programa de Pós-Graduação em Tecnologias Educacionais em Rede - PPGTER
Editoria Técnica do PPGTER
Universidade Federal de Santa Maria
Av. Roraima n. 1000
Centro de Educação, Prédio 16, sala 3146
Santa Maria – RS – CEP 97105-900

Fone / FAX: 55 3220 9414
ppgter@ufsm.br
edtec.ppgter@gmail.com

ISSN: 2675-0309

Relatórios Técnicos do Programa de Pós-Graduação em Tecnologias Educacionais em Rede / Programa de Pós-Graduação em Tecnologias Educacionais em Rede, Universidade Federal de Santa Maria. – Vol. 1. n. 1 (2019) Jan/Jul. – Santa Maria: PPGTER/UFSM, 2019.

Periodicidade semestral.

1. Tecnologia Educacional. 2. Desenvolvimento de Tecnologias Educacionais. 3. Gestão de Tecnologias Educacionais. I. Universidade Federal de Santa Maria. Programa de Pós-Graduação em Tecnologias Educacionais em Rede.

Resumo

Este relatório técnico tem como objetivo apresentar o produto final da pesquisa intitulada "**Ensino de Programação Orientada a Objetos na Educação Profissional por Meio do Desenvolvimento de Jogos Apoiado Pelo Ambiente Greenfoot**" (RICHTER, 2018), desenvolvida como parte da dissertação de mestrado apresentada ao Programa de Pós-Graduação em Tecnologias Educacionais em Rede, na linha de pesquisa Desenvolvimento de Tecnologias Educacionais em Rede. As Unidades de Estudo desenvolvidas durante a pesquisa podem ser reutilizadas, bem como adaptadas de acordo com os objetivos específicos de cada professor. A pesquisa foi conduzida pelo mestrando Cleitom José Richter sob orientação da professora Giliane Bernardi e aplicada junto ao curso Técnico em Informática Integrado ao Ensino Médio do Instituto Federal Farroupilha - Campus Santo Augusto.

Referências

RICHTER, Cleitom José. **Ensino de Programação Orientada a Objetos na Educação Profissional por Meio do Desenvolvimento de Jogos Apoiado Pelo Ambiente Greenfoot.** 2018. 205 p. Dissertação (Mestrado Profissional em Tecnologias Educacionais em Rede) – Universidade Federal de Santa Maria, Santa Maria, 2018.

APÊNDICE A

Aprendendo orientação à Objetos com Greenfoot

Aprendendo orientação a objetos com Greenfoot

- Programação Orientada a Objetos
- Desenvolvimento de Jogos
- IDE Greenfoot
- Unidades de Estudo
- Mapas Conceituais
- Ciclo de Aprendizagem de Kolb

Desenvolvimento

CLEITOM RICHTER

cleitom.richter@iffarroupilha.edu.br

Orientação

GILIANE BERNARDI

giliane@inf.ufsm.br

Universidade Federal de Santa Maria

SEDES SAPIENTIÆ

1960

PPGTER

Programa de Pós-Graduação em Tecnologias Educacionais em Rede

http://www.ppgter.ufsm.br

Cleitom José Richter

Aprendendo Orientação a Objetos com Greenfoot

Santa Maria - RS
Agosto 2018

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 tornou-se 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, este trabalho tem o objetivo de apresentar o produto final da pesquisa intitulada "**Ensino de Programação Orientada a Objetos na Educação Profissional por Meio do Desenvolvimento de Jogos Apoiado Pelo Ambiente Greenfoot**", conduzida pelo mestrando Cleitom José Richter sob orientação da professora Giliane Bernardi.

Tal pesquisa foi motivada pela verificação de dificuldades dos estudantes em assimilar os conceitos envolvidos na programação de computadores e, sobretudo, no que se refere ao paradigma de Programação Orientada a Objetos (POO). Assim, a proposta desenvolvida na referida pesquisa, buscou encontrar uma estratégia de ensino e aprendizagem de POO capaz de amenizar as dificuldades encontradas pelos estudantes e professores.

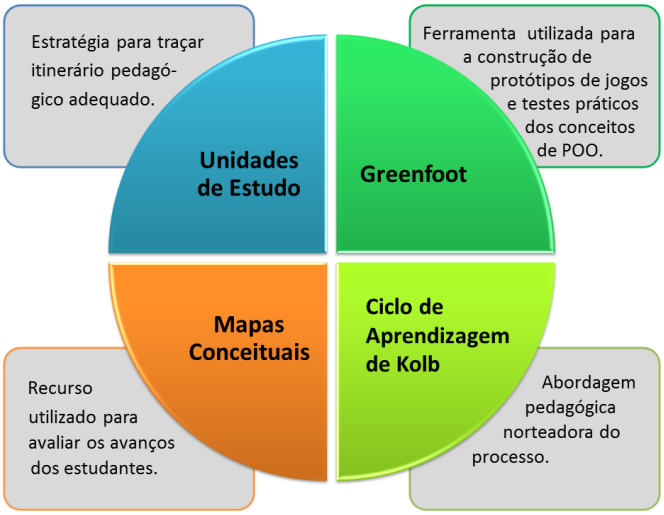
O desenvolvimento da proposta foi norteado pela estruturação de Unidades de Estudo abordando o desenvolvimento de protótipos de jogos com a finalidade intrínseca de proporcionar condições para a aprendizagem dos conceitos de orientação a objetos, bem como, promover melhoria nas intervenções didáticas no ensino de programação de computadores. Nas referidas unidades de estudo, os protótipos de jogos são criados com apoio da IDE Greenfoot, a qual foi projetada, especificamente, para auxiliar no ensino de POO a partir do desenvolvimento de jogos.

Para verificar os avanços dos estudantes, utilizou-se a construção de Mapas Conceituais, pois são caracterizados como um recurso valioso para organizar e avaliar a estrutura cognitiva dos estudantes, seja pelo ponto de vista qualitativo ou quantitativo, conforme ilustraremos nas próximas seções.

O embasamento pedagógico da prática é norteado pela proposta do Ciclo de Aprendizagem de Kolb (KOLB, 1984), o qual considera que a aprendizagem ocorre por meio de um ciclo contínuo composto de quatro estágios: Experiência Concreta (agir); Observação Reflexiva (refletir); Conceitualização Abstrata (conceitualizar) e Experimentação Ativa (aplicar).

Assim, os elementos utilizados na estratégia de ensino de POO seguem a organização ilustrada pela Figura 1.

Figura 1 - Materiais utilizados no desenvolvimento da pesquisa



Fonte: do Autor.

Cada um dos materiais utilizados serão descritos nas próximas seções, bem como são apresentadas as Unidades de Estudo desenvolvidas durante a pesquisa, que podem ser reutilizadas, bem como adaptadas de acordo com os objetivos específicos de cada professor.

Aprendendo orientação a objetos com Greenfoot

2 IDE GREENFOOT

Nessa seção apresentamos a IDE Greenfoot, que é a ferramenta escolhida para dar suporte à criação dos protótipos de jogos com a finalidade de ensinar e aprender os conceitos de POO.

Greenfoot é uma IDE[1] 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 que estudantes de computação uma perspectiva mais lúdica acerca da utilização do paradigma de POO por meio do Greenfoot.

Em essência, o ambiente 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 2).

Figura 2 - Logomarca Greenfoot

Fonte: Greenroon (2017).

1 IDE: do inglês Integrated Development Environment ou Ambiente de Desenvolvimento Integrado.

Trata-se de software de desenvolvimento com bons recursos de programação, sendo que o código fonte da sua IDE encontra-se sob licença GNU[2] General Public Licence versão 2, com exceção do classpath.[3] Desse modo, no site oficial da ferramenta (www.greenfoot.org) é possível realizar o download do arquivo de instalação e do seu código fonte. No site, também estão disponíveis tutoriais, suporte, cenários e exemplos. 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:

1) **Actor** - Contém os métodos disponíveis para os atores do jogo.
2) **Color** – Usada para preencher cores na tela.
3) **Font** - A Fonte pode ser usada para escrever texto na tela;
4) **Greenfoot** – Usado para se comunicar com o Ambiente Greenfoot em si;
5) **GreenfootImage** – contém métodos para apresentação e manipulação de imagens.
6) **GreenfootSound** - contém métodos para apresentação e manipulação de sons;
7) **MouseInfo** – Métodos para capturar eventos do Mouse;

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

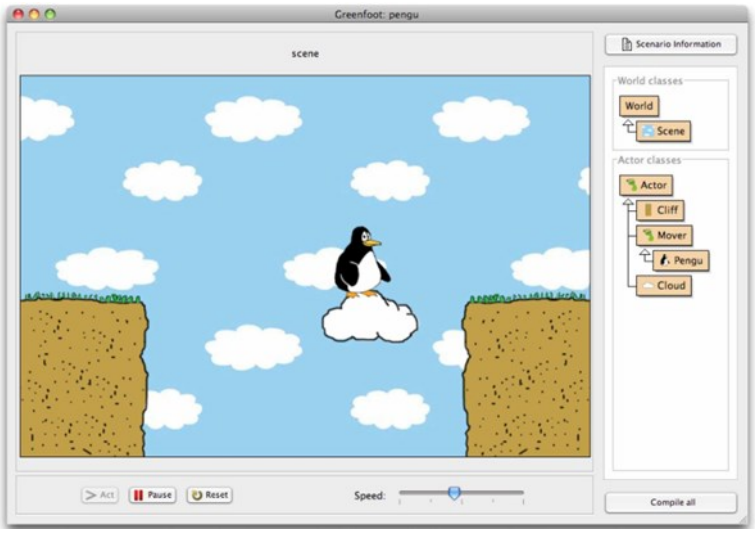
3 Classpath - É uma variável de ambiente utilizada pela linguagem JAVA para armazenar informações acerca das localizações dos arquivos e bibliotecas necessários para a compilação e execução de um programa criado nessa linguagem. Dessa forma, essa variável desempenha papel fundamental no funcionamento de qualquer software desenvolvido em JAVA.

8) **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;

9) **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 3 ilustra a tela principal do Greenfoot.

Figura 3 - Tela principal do Greenfoot

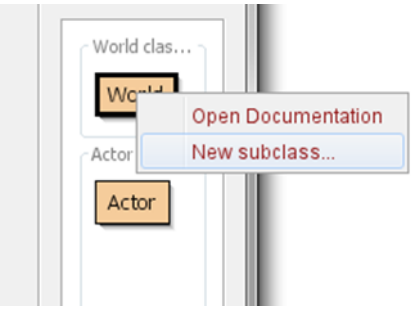


Fonte: Kölling (2010).

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 4).

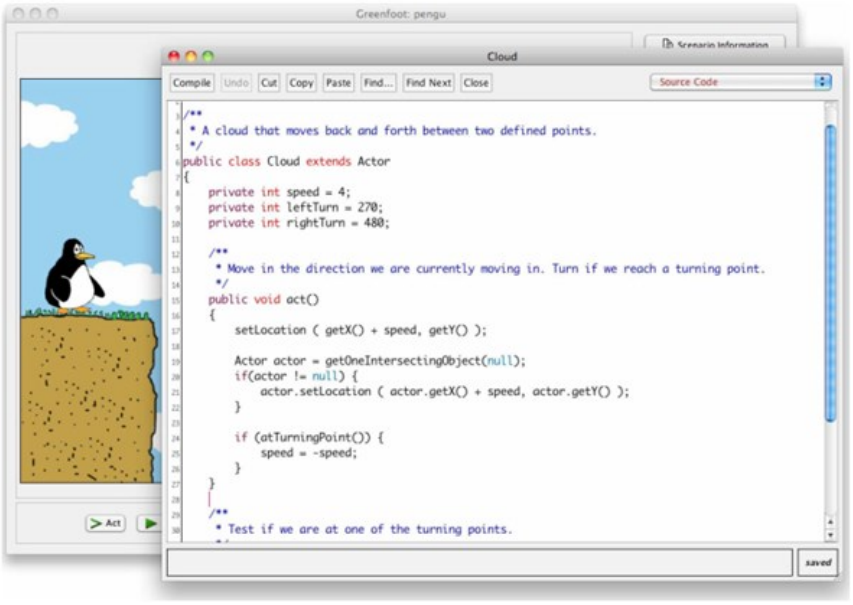
Figura 4 - Criando subclasse de World



Fonte: do Autor.

Para além da tela principal é possível acessar o código fonte das classes criadas (Figura 5), 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 5 - 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 têm 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:

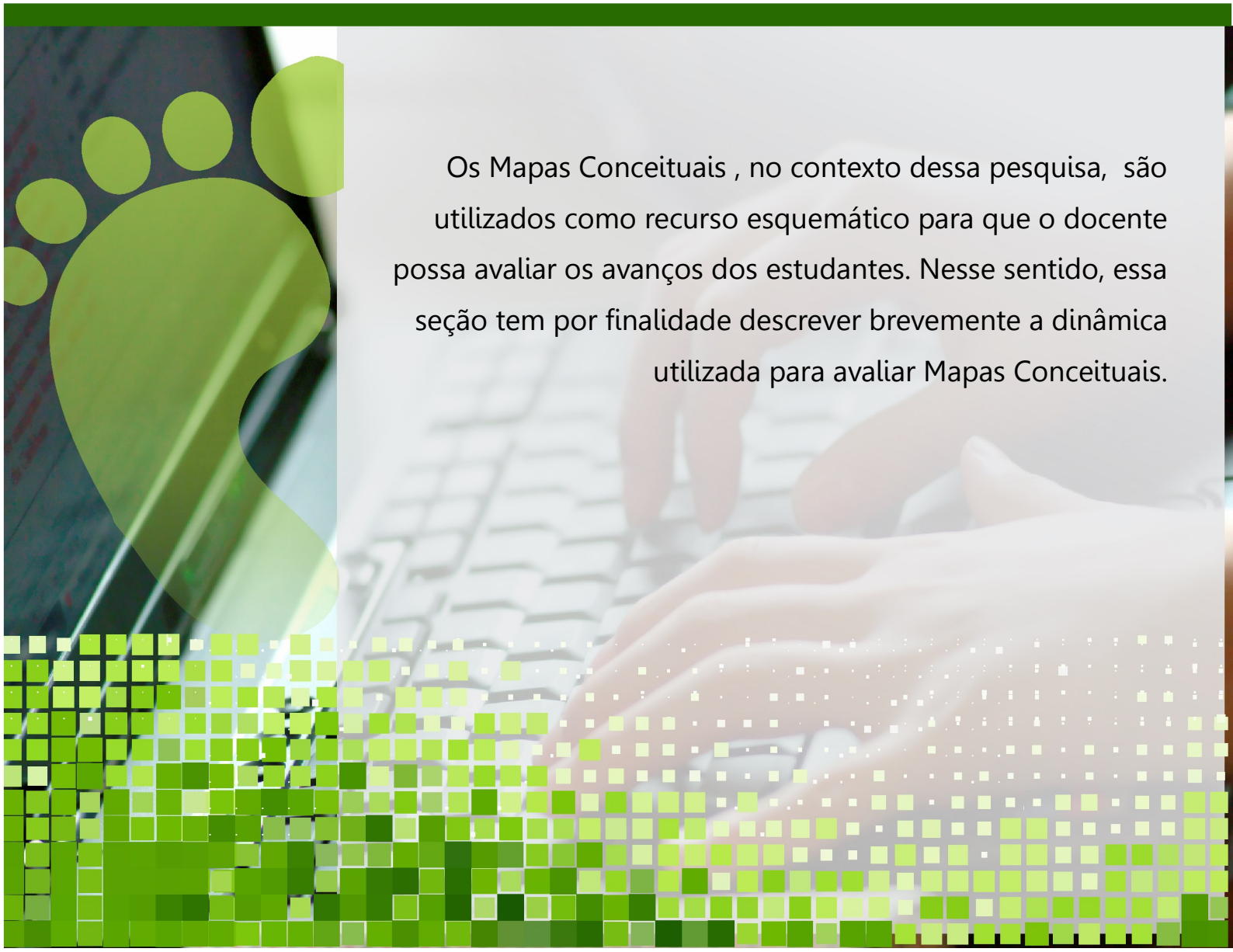
a) Um programa consiste em um conjunto de classes;
b) A partir das Classes, podemos criar objetos;
c) Múltiplos objetos podem ser criados a partir de uma classe;
d) Todos os objetos da mesma classe oferecem os mesmos métodos e têm os mesmos campos;
e) Cada objeto possui seus próprios valores para seus 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.

Aprendendo orientação a objetos com Greenfoot

3 MAPAS CONCEITUAIS



Os Mapas Conceituais , no contexto dessa pesquisa, são utilizados como recurso esquemático para que o docente possa avaliar os avanços dos estudantes. Nesse sentido, essa seção tem por finalidade descrever brevemente a dinâmica utilizada para avaliar Mapas Conceituais.

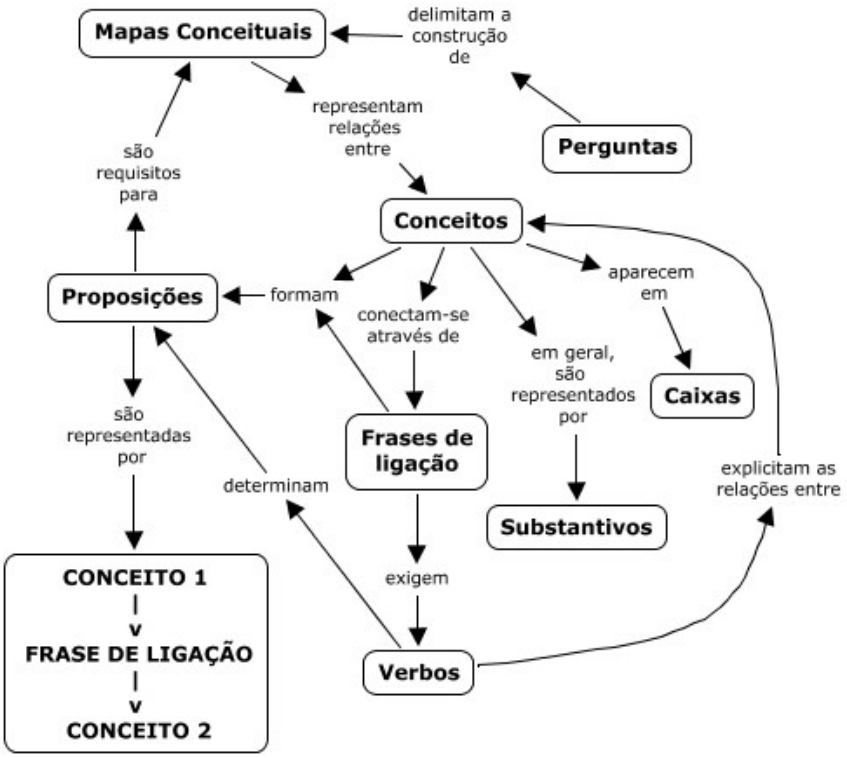
Mapa conceitual é um recurso para organizar e representar a estrutura cognitiva do indivíduo. Pode ser utilizado para ilustrar de maneira gráfica as relações entre os conceitos de um determinado assunto. Nesse caso, mapas conceituais operam como uma importante ferramenta para que o aprendiz organize 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 individuo (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.

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 6.

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.

Figura 6 - Exemplo de Mapa Conceitual



Fonte: CEMED (2010).

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:

> As *proposições*, isto é, os conceitos com as palavras de ligação apropriadas que nos indicarão as relações válidas ou equivocadas.
>
> A *hierarquização*, sempre no sentido de que os conceitos mais gerais incluam mais específicos.
>
> As *relações cruzadas*, que mostram as relações entre conceitos pertencentes a diferentes partes do mapa conceitual.
>
> 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 desse recurso 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 nas teoria de Ausubel. A Tabela 1 ilustra os critérios e respectivas pontuações.

Tabela 1 - Critérios avaliação dos mapas conceituais

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

Fonte: Martins et al (2009) com adaptações.

Aprendendo orientação a objetos com Greenfoot

4 UNIDADES DE ESTUDO

Unidades de Estudo, segundo Filatro e Cairo (2015), fazem referência ao desenvolvimento de um tipo de Metamodelo Educacional, cuidadosamente elaborado para que atenda as necessidades de aprendizagens dos estudantes. No caso desta pesquisa, as Unidades de Estudo compreendem a estratégia para traçar o itinerário pedagógico, o qual, nesse caso, envolve o desenvolvimento de jogos (com ênfase nos conceitos de POO) e questionários .

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 seus 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.

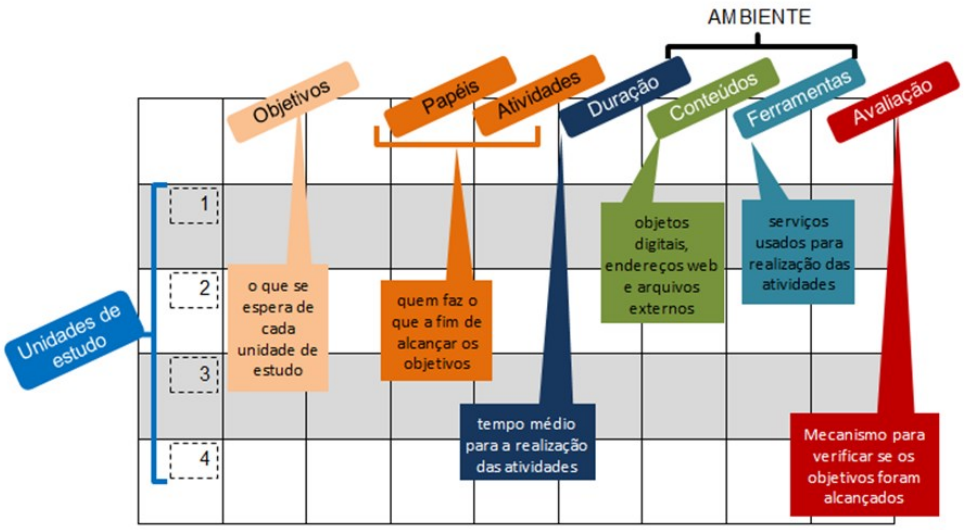
Segundo as autoras, o planejamento educacional resulta em produtos, que podem ser materializados na forma de: livros impressos e digitais, objetos de aprendizagem, podcasts, vídeos, infográficos, etc., sendo que cada produto deve ser elaborado com base na metodologia da gestão de projetos (FILATRO e 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.

Assim, para a construção de um planejamento adequado e eficiente, Filatro e Cairo (2015) sugerem a utilização de um "*Metamodelo Educacional*" 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 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).

A matriz capaz de agregar uma estrutura que contemple a proposta do metamodelo educacional é ilustrada pela Figura 7.

Figura 7 - Matriz de planejamento baseada em metamodelo educacional



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

A Figura 7 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.

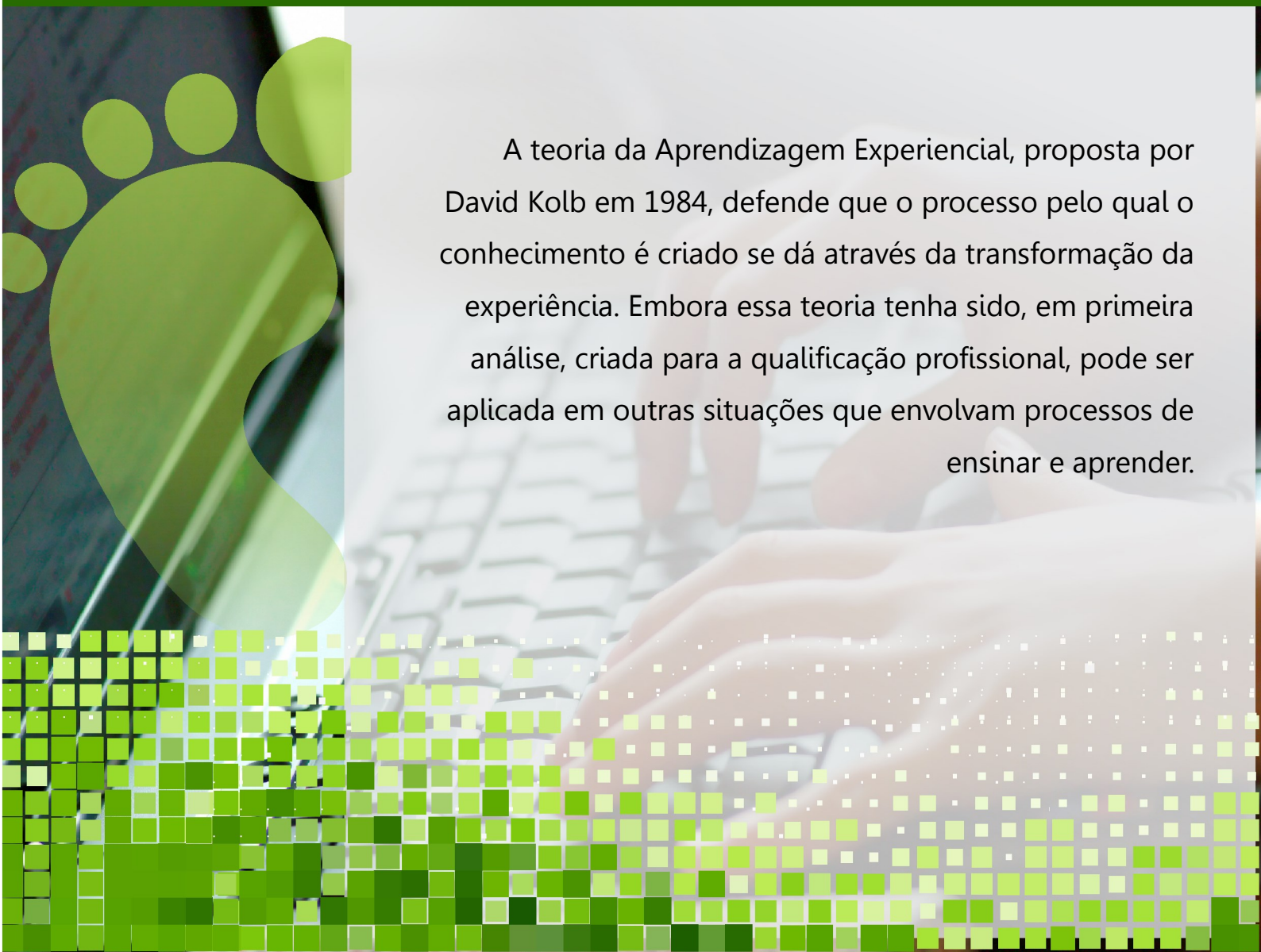
Os "papéis" representam outro elemento especial do metamodelo, pois, diferente do que costumamos verificar no plano de aula, existe certa dinamicidade nesse quesito, visto que os atores poderão desempenhar diferentes papéis dependendo da proposta.

> 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).

Aprendendo orientação a objetos com Greenfoot

5 CICLO DE APRENDIZAGEM DE KOLB



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.

O Ciclo de Aprendizagem de Kolb, foi desenvolvido na década de 1980 por David Kolb e é parte integrante da sua teoria da aprendizagem experiencial (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 2 descreve cada um dos estágios desse círculo, elencando as principais ações características de cada um.

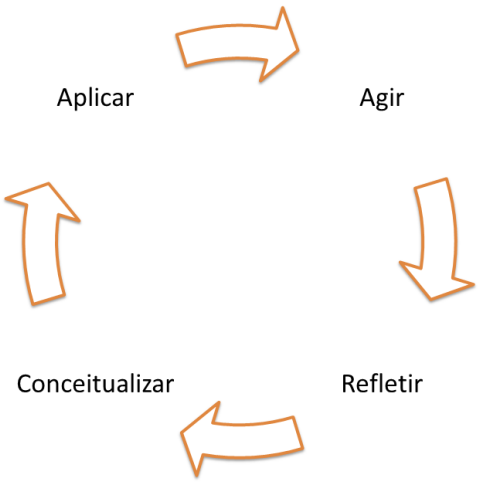
Tabela 2 - Estágios do processo de aprendizagem segundo David Kolb

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 (Figura 8).

Figura 8 - 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 oferecerá experimentações e nova experiência concreta.

Aprendendo orientação a objetos com Greenfoot

6 UNIDADE DE ESTUDO I

Construção do projeto "Travessia"

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.

APRESENTAÇÃO DA UNIDADE DE ESTUDO I

A **Unidade de Estudo I** 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 apoio à atividade. 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.

Unidade de Estudo I

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

6 horas aula;

4 Conteúdos

{Abstração, Classe, Objeto, Atributo, Método, Construtor, Instanciação, Mensagem}

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

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

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

Banco de imagens: https://drive.google.com/open?id=14Jeq-i5YzUZw7csRypQekPzXoDAn8DLS

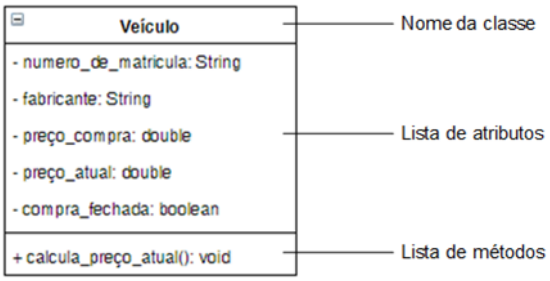
5 Atividades

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. A Figura 9 ilustra a representação UML de uma classe hipotética 'Veículo'.

Figura 9 - Representação de uma classe



Assim, como se pode verificar na Figura 9, 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 computa-

cional (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 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

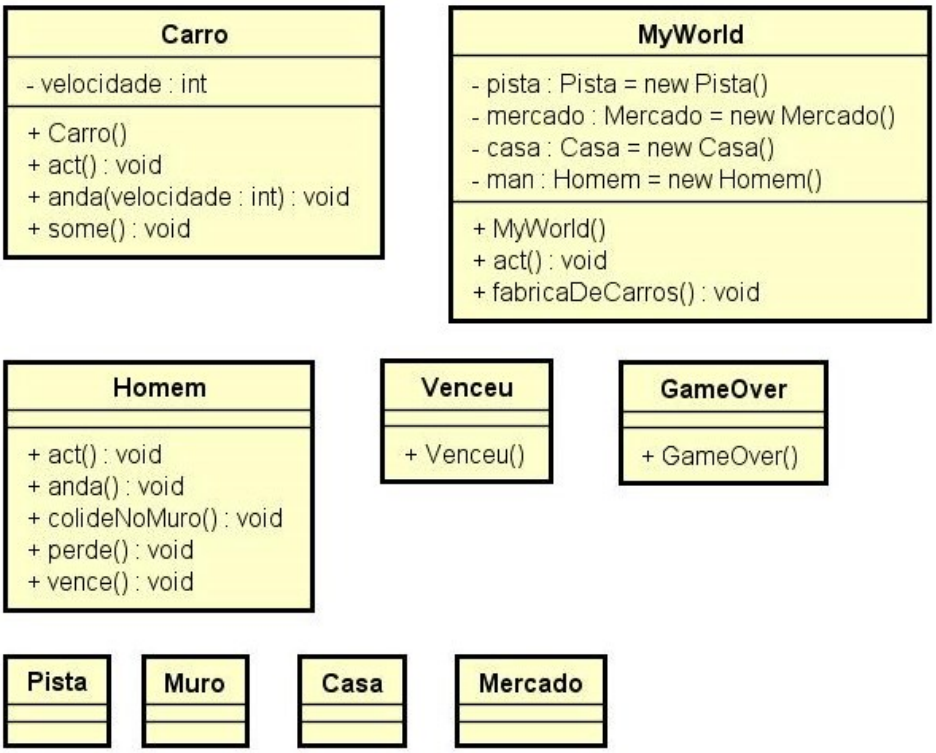
Utilizando o Greenfoot, construir um protótipo de jogo conforme ilustrado pela Figura 10, Vídeo 1 e Modelagem do Projeto Travessia (Figura 11). Na sequência responda às questões do item 7 Avaliação.

Figura 10 - Prospecto do Projeto Travessia

Vídeo 1 - Vídeo do Projeto Travessia

Link: https://youtu.be/3mkCBIwaCH4

Figura 11 - Modelagem do Projeto Travessia



5.3 Descrição das classes do projeto

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. man tipo Homem;

- Métodos:
  a. **MyWorld()** (Construtor): adiciona os objetos pista, mercado, casa e man nas coordenadas adequadas [addObject(objeto, x, y)] – adiciona ainda, os muros nas laterais da lista para impedir que o objeto man 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;`];

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, porém nesse caso, é necessário apenas chamar o método *anda(int velocidade)*, passando por parâmetro a velocidade do objeto [this.velocidade];

b. **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];

c. **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 "*manD.png*" e "*manE.png*";

c. **colideNoMuro():** Esse método tem a finalidade de impedir que o objeto *man* do tipo *Homem* possa passar por cima dos objetos do tipo *Muro*, nesse caso, utiliza-se o método da classe Actor getOneIntersectingObject(class<?>)

Figura 12 - Exemplo de código colideNoMuro()

```
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);
    }
}
}
```

para verificar se o objeto *man* interceptou outo objeto da classe *Muro* e, caso verdadeiro, verificar se está abaixo ou acima dele, nesse caso modificar a posição do objeto *man* (em 40px) para que não consiga atravessar o muro.

d. **perde():** verifica se o objeto *man* 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 *man* 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)].

Figura 13 - Exemplo de construtor da classe GameOver

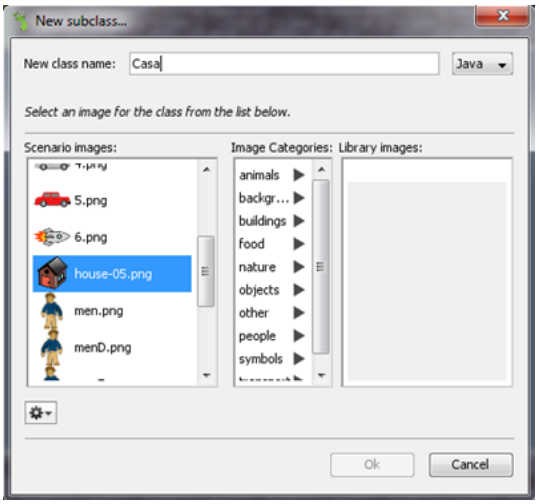
```
public GameOver(){
    GreenfootImage texto;
    texto = new GreenfootImage("Game Over", 60, Color.RED, new Color(0,0,0,0) , Color.BLACK);
    setImage(texto);
}
```

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 ilustrado pela Figura 14. 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.

Figura 14 - Criando classe no Greenfoot



DESAFIO

Vamos nos desafiar no processo de aprendizagem de POO. Pense em alguma funcionalidade extra para facilitar a travessia do objeto *man*. 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.

ENCONTRE AJUDA EXTRA

⇒ Site oficial do Greenfoot www.greenfoot.org dispõe de tutoriais, vídeos e exemplos para *download*;

⇒ Canal do "Brasilia Java Users Group – DFJUG" no youtube, especialmente nas playlists: "*Greenfoot Five Minutes - Mastering the API* " e "*GreenLabs – Laboratório de Jogos do DFJUG* ".

6 Ferramentas

IDE Greenfoot.

7 Avaliação

7.1 Questões sugeridas

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 recepção de mensagens, em que ponto do desenvolvimento do projeto "Travessia" isso fica evidente? Como isso acontece?

Aprendendo orientação a objetos com Greenfoot

7 UNIDADE DE ESTUDO II

Construção do projeto “Simulador”

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;

APRESENTAÇÃO DA UNIDADE DE ESTUDO II

A **Unidade de Estudo II** tem como objetivo 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).

Inicialmente são disponibilizados 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.

Unidade de Estudo II

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

9 horas aula;

4 Conteúdos

{Encapsulamento e Métodos Acessores}

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

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

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

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 14 - 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;
    }
```

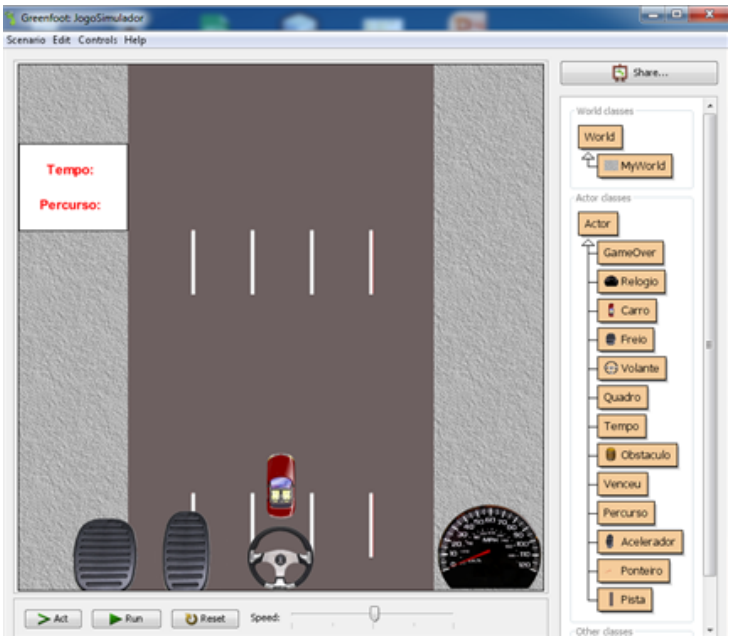
Fonte: Carvalho e Teixeira (2012, p.53).

> 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)..

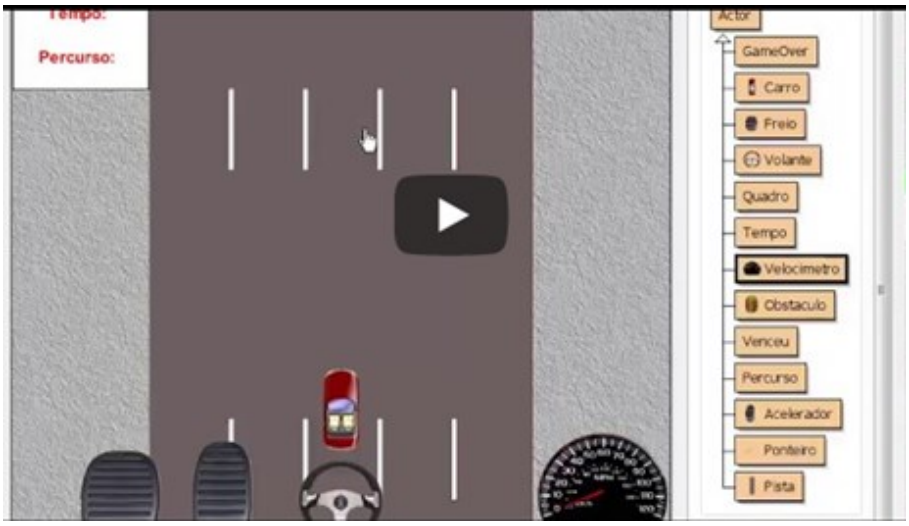
5.2 Atividade proposta

Passo 1: Utilizando o Greenfoot, construir o projeto "Simulador" conforme Figura 15 e Vídeo 2 abaixo:

Figura 15 - Prospecto do Projeto Simulador



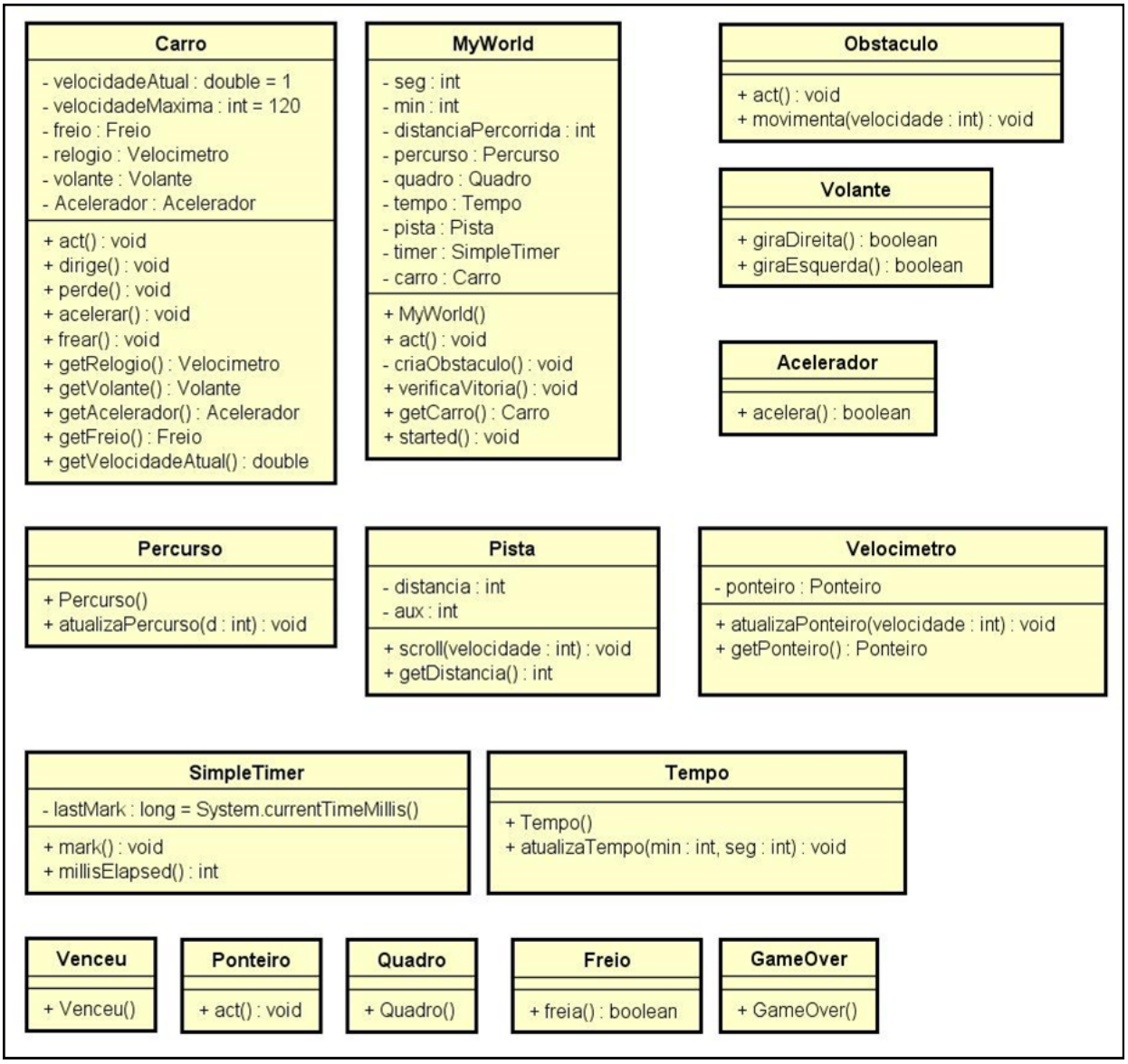
Vídeo 2 - Vídeo do Projeto simulador



Link: https://youtu.be/SXUI2m1Rfy4

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

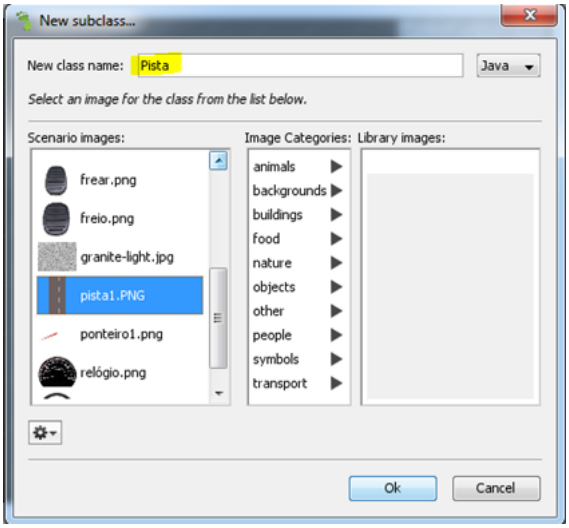
Figura 16 - Modelagem do Projeto Simulador



5.3) Descrição das classes

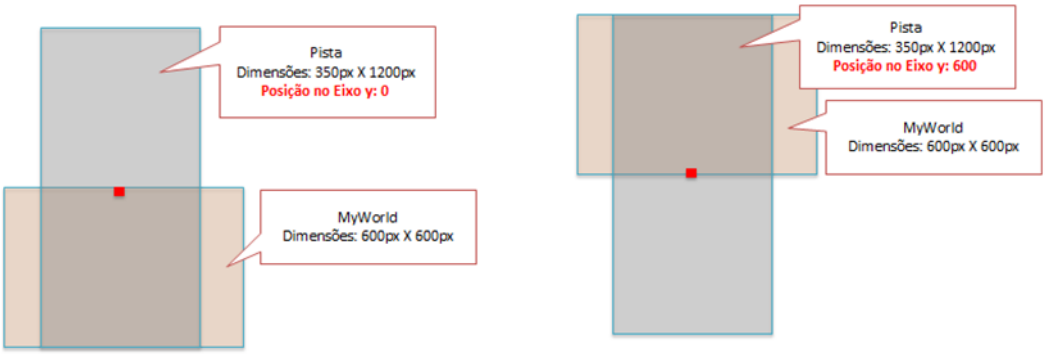
I. Descrição da classe "Pista": A classe pista tem apenas os atributos 'distancia' e 'aux' e os métodos 'scroll()' e 'getDistancia()'. A 'image' pode ser definida no momento da criação da classe (Figura 17).

Figura 17 - Criando classe Pista



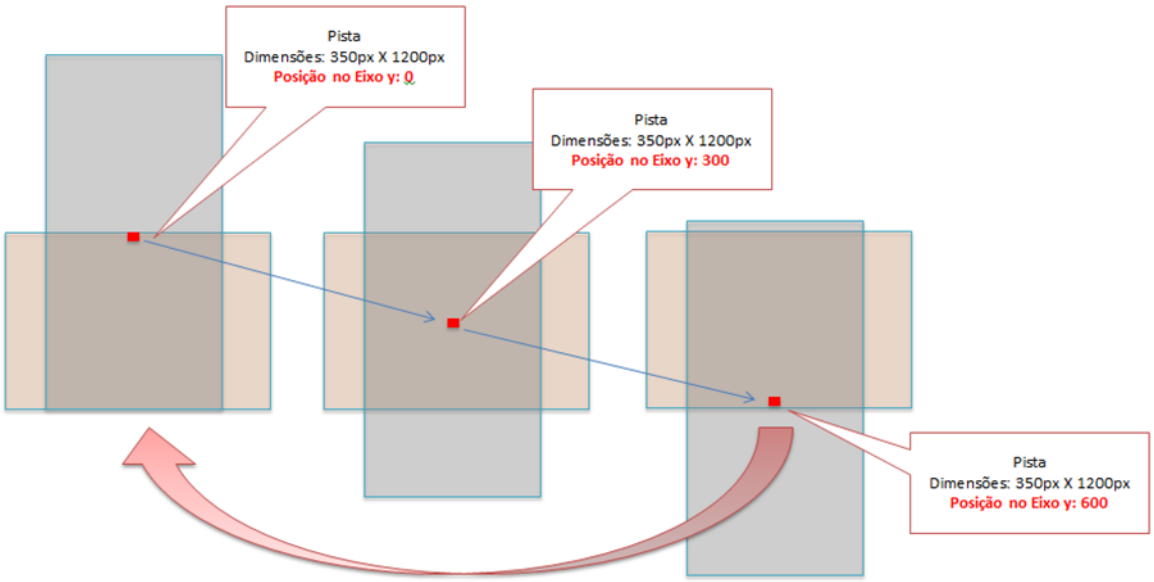
a. **scroll():** A pista tem 1200px de altura, enquanto que o cenário (MyWorld) tem 600px. É importante observar que o Greenfoot leva em consideração o meio do objeto para indicar a sua localização no Plano Cartesiano, veja esquema (Figura 18):

Figura 18 - 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 19.

Figura 19 - Funcionamento do método scroll



b. **getDistancia():** retorna o atributo distancia.

II. Descrição da classe "Freio": Na classe Freio tem o atributo "image"(herdado de Actor) deve ser alimentado com o valor "images/freio.png" que se encontra no banco de imagens. O método freia deve ser implementado conforme descrição abaixo:

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");

III. Descrição da classe "Ponteiro": A classe Ponteiro tem apenas atributos herdados de Actor, nesse caso "image" deve ser alimentado com o valor "images/ponteiro.png" que se encontra no banco de imagens.

IV. Descrição da classe "Velocimetro": Na classe *Velocimetro* o atributo "image" (GreenfootImage) deve ser alimentado com o valor "images/ relogio.png". Possui também o atributo ponteiro (tipo *Ponteiro*). A classe possui também os métodos *atualizaPonteiro(int)* e *getPonteiro():*

a. **atualizaPonteiro(int):** esse método não tem retorno (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".

V. 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 usando *setRotation(int)*) . 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);

VI. Descrição da classe "Acelerador": possui apenas atributos herdados de *Actor*, sendo que "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");

VII. 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. **getVelocimetro():** retorna o atributo velocímetro;
  g. **getAcelerador():** retorna o atributo acelerador;
  h. **getVolante():** retorna o atributo volante;
  i. **getFreio():** retorna o atributo freio;
  j. **getVelocidadeAtual():** retorna o atributo velocidadeAtual.

VIII. 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.

- **Métodos**
  a. **MyWorld() (construtor):** adiciona os objetos [*addObject (_objeto_, _x_,_y_)*] *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* **que são atributos em *carro***. E através do *velocímetro* que está no carro, adiciona o *ponteiro* (todos em coordenadas adequadas conforme Figura 15).
  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(). O método atualizaPercuros() do objeto 'percurso' deve ser invocado no act() para que a distância percorrida seja atualizada [percurso.atualizaPercurso(int)], passando por parâmetro a distância percorrida pela pista [pista.getDistancia()] - importante dividir a distância percorrida por 20, pois caso contrário o valor avança muito rápido. 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 = timer.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.

IX. Descrição da classe Obstáculo: 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());*].

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;

XI. 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)].

XII. 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)*].

XIII. 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)*].

IV. Descrição da classe Venceu: não possui atributos. Tem a finalidade de criar objeto contendo o texto que indica vitória do personagem principal.

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)*].

V. 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.

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)*].

Figura 20 - Construtor da classe Quadro

```
public Quadro(){
    GreenfootImage quadro = new GreenfootImage(125,100 );
    quadro.setColor(Color.BLACK);
    quadro.drawRect(0, 0, quadro.getWidth()-1, quadro.getHeight()-1);
    quadro.setColor(Color.WHITE);
    quadro.fillRect(1, 1, quadro.getWidth()-2, quadro.getHeight()-2);
    setImage(quadro);
}
```

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.

ENCONTRE AJUDA EXTRA

⇒ Site oficial do Greenfoot www.greenfoot.org dispõe de tutoriais, vídeos e exemplos para *download*;

⇒ Canal do "Brasilia Java Users Group – DFJUG" no youtube, especialmente nas playlists: "*Greenfoot Five Minutes - Mastering the API*" e "*GreenLabs – Laboratório de Jogos do DFJUG*".

6 Ferramentas

IDE Greenfoot.

7 Avaliação

7.1 Questões sugeridas para discussão

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 identificá-los no projeto?

Aprendendo orientação a objetos com Greenfoot

8 UNIDADE DE ESTUDO III

Construção do projeto “Colheita das Maçãs”

Objetivos

Ratificar os conceitos de classe, objeto, método, atributo, instanciação, construtor, envio de mensagem, encapsulamento;

Compreender o que é herança e sua relação com o polimorfismo;

Perceber como os conceitos de PPO se relacionam a fim de possibilitar a construção eficiente de um produto de software.

APRESENTAÇÃO DA UNIDADE DE ESTUDO III

O objetivo principal da **Unidade de Estudo III** é de proporcionar ao estudante a percepção de como os conceitos de POO se relacionam e, dessa maneira possibilitam a construção eficiente de um produto de software. Assim, esta unidade de estudo busca, através das dinâmicas proposta, ratificar 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.

Unidade de Estudo III

1 Objetivos

- Ratificar os conceitos de classe, objeto, método, atributo, instanciação, construtor, envio de mensagem, encapsulamento;
- Compreender o que é herança e sua relação com o 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.

3 Duração

12 horas aula;

4 Conteúdos

{União de todos os conceitos}

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

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

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

Banco de imagens: https://drive.google.com/drive/folders/1_He062HnfW2Csv0-cBvkXm6UQhG-6nlg?usp=sharing

5 Atividades

5.1 Considere as premissas básicas relacionadas à Programação Orientada a Objetos

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

Utilizando o Greenfoot, vamos construir na íntegra o projeto "Colheita das Maçãs" conforme Descrição do Jogo, Figura 21 e Vídeo 3.

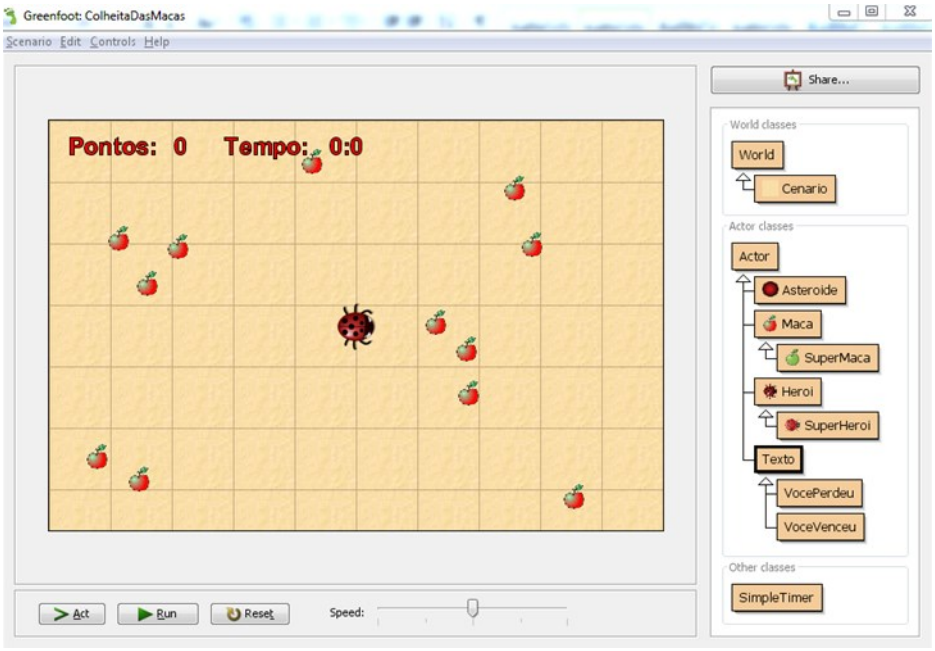
Descrição do Jogo "Colheita das Maçãs"

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.

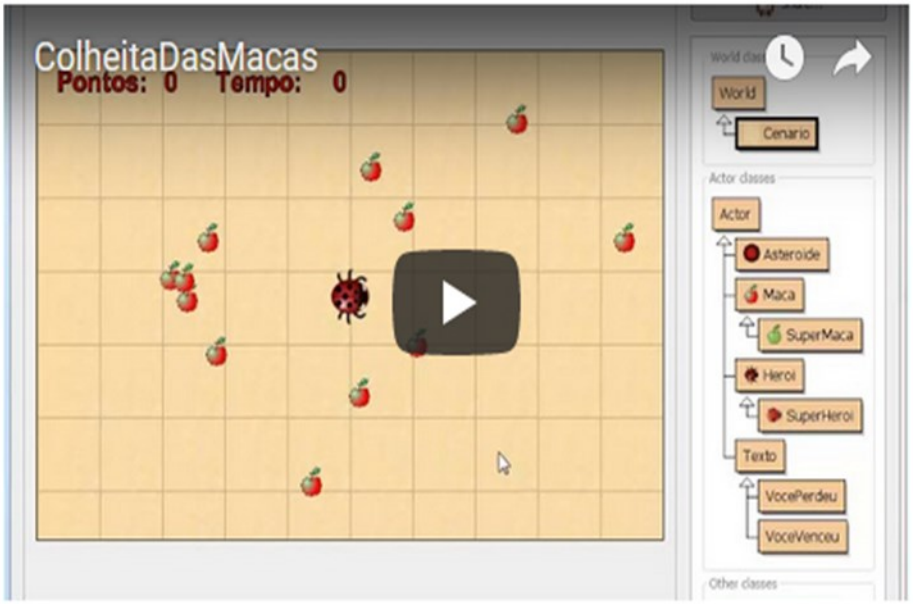
Como desafio, o personagem principal precisa desviar-se de uma chuva de meteoros que caem aleatoriamente no cenário, pois caso algum se choque com o herói o jogo acaba e ele perde.

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 21 - Prospecto do Projeto Colheita das Maçãs



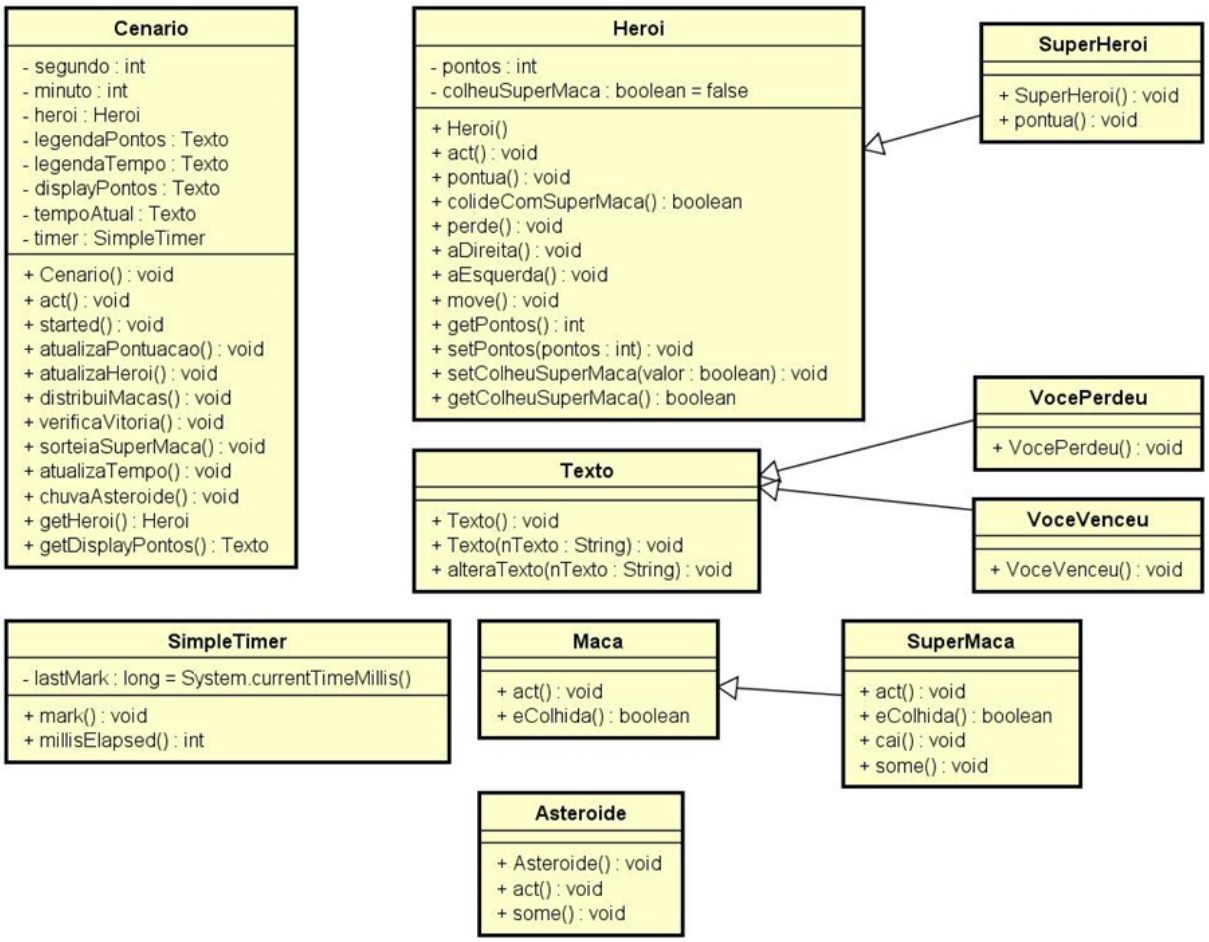
Vídeo 3 - Vídeo do Projeto Colheita das Maçãs



Fonte: https://youtu.be/cVZZheoeM_E

Realize a implementação conforme modelagem UML e descrição abaixo.

Figura 22 - Modelagem (Diagrama de classes) Projeto Colheita das Maçãs



5.3) Descrição das classes

I. Descrição da classe "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 herói;

l. **getDisplayPontos():** retorna o atributo displayPontos.

II. Descrição da Classe "Heroi"

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 ():** retorna o 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;* sui apenas o comando de atribuição da imagem *ladybug_02.png* para o objeto [*setImage(String imagem)];*

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 "VocePerdeu"

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", 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. **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*.

b. **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()*;

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.

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.

ENCONTRE AJUDA EXTRA

⇒ Site oficial do Greenfoot www.greenfoot.org dispõe de tutoriais, vídeos e exemplos para *download*;

⇒ Canal do "Brasilia Java Users Group – DFJUG" no youtube, especialmente nas playlists: "*Greenfoot Five Minutes - Mastering the API*" e "*GreenLabs – Laboratório de Jogos do DFJUG*".

6 Ferramentas

IDE Greenfoot.

7 Avaliação

7.1 Sugestão de dinâmica para o grupo de estudantes

Atividade em grupo: 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, 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.

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.

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.

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

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.

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.

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.

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

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.

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

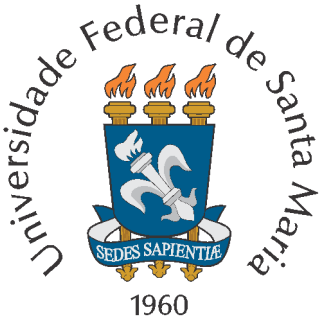
REFERÊNCIAS

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.

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.

Aprendendo orientação a objetos com Greenfoot



PPGTER
Programa de Pós-Graduação em
Tecnologias Educacionais em Rede