UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS - GRADUAÇÃO EM ENSINO DE FÍSICA
MESTRADO NACIONAL PROFISSIONAL EM ENSINO DE FÍSICA

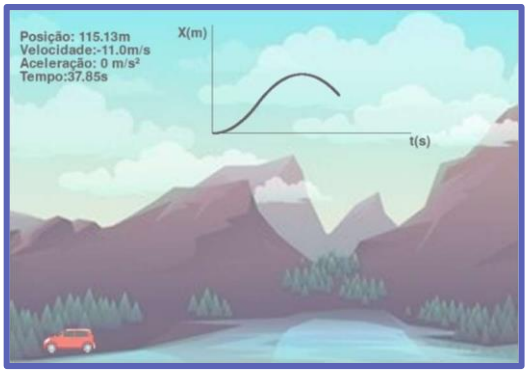
PRODUTO EDUCACIONAL

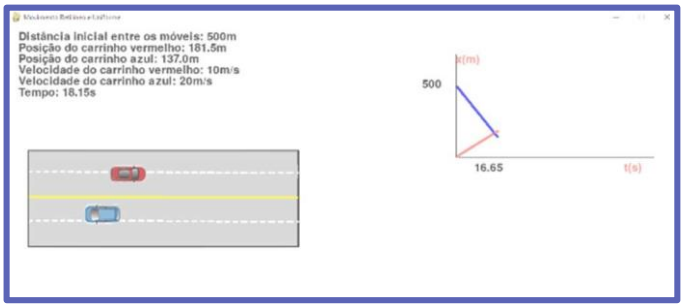
MODELAGEM COMPUTACIONAL DE QUESTÕES DE CINEMÁTICA EM PYTHON

LEONARDO BARRETO BAPTISTA
MARIA LÚCIA DE MORAES COSTA

Belém, Pará, Brasil
2022

MODELAGEM COMPUTACIONAL DE QUESTÕES DE CINEMÁTICA EM PYTHON





Modelos Físicos Construcionismo Pensamento Computacional

Leonardo Barreto Baptista

Maria Lúcia de Moraes Costa

Belém - 2022

PRODUTO EDUCACIONAL

RESUMO

Este trabalho apresenta uma sequência didática para *Modelagem Computacional de Questões de Cinemática, em Python,* como um recurso de aprendizagem ativa e exercício do Pensamento Computacional, associado à Modelagem Científica no Ensino de Física, com o objetivo de promover a compreensão dos estudantes sobre os processos envolvidos na representação dos fenômenos físicos. Disponibilizamos, neste material, videoaulas sobre conceitos básicos de Linguagem Python e dois exemplos de construção de modelos computacionais, um de movimento retilíneo uniforme e outro, de movimento retilíneo uniformemente variado. Propomos exercícios, com ênfase nos conceitos físicos, que servirão de suporte para a compreensão da modelagem computacional e análise de domínio de validade dos modelos teóricos apresentados no livro didático, utilizado pelo professor. O principal referencial teórico deste trabalho é a Teoria Construcionista, de Seymour Papert, que propõe que o aluno aprende melhor através de uma construção explícita, como é o caso da programação computacional.

Palavras-chave: Ensino de Física, Modelagem Científica, Teoria Construcionista, Pensamento Computacional, Python, Cinemática.

Belém, Pará, Brasil

2022

Sumário

Apresentação 8

1. Fundamentação Teórica 10

1.1 Por que ensinar Modelagem Científica no Ensino Médio? 10

1.2 A importância da construção na aprendizagem 14

1.3 O pensamento computacional 16

2. Criando animações com o Pygame 20

3. Metodologia 24

3.1 Estudo do movimento retilíneo e uniforme 25

3.1.1 Resumo Teórico 25

3.1.2 Atividade teórica proposta 28

3.1.3 Atividade de modelagem computacional 31

3.2 Estudo do movimento retilíneo uniformemente variado 36

3.2.1 Resumo Teórico 36

3.2.2 Atividade teórica proposta 40

3.2.3 Atividade de modelagem computacional 42

3.3 Estudo do lançamento vertical e queda livre 48

3.3.1 Resumo Teórico 48

3.3.2 Atividade teórica 50

3.3.3 Atividade de modelagem computacional 53

3.4 Estudo do lançamento oblíquo 54

3.4.1 Resumo Teórico 54

3.4.2 Atividade teórica 58

3.4.3 Atividade de modelagem computacional 60

4. Considerações finais 62

Referências 63

Apêndice A - A Linguagem Python e o Pygame 65

A.1. A Linguagem Python 65

A.1.1 Origem 65

A.1.2 Vantagens 65

A.1.3 Fundamentos **66**
A.1.4 Sintaxe **66**
A.1.5 Interpretador Python **68**
A.1.6 Ambientes de Desenvolvimento de Integrado **68**
A.1.7 Variáveis **69**
A.1.8 Tipos de Dados em Python **70**
A.1.9 Coleções **72**
A.1.10 Operadores **74**
A.1.11 Estruturas Condicionais **77**
A.1.12 Estruturas de Repetição **77**
A.1.13 Funções **80**
A.2. O Pygame **81**
A.2.1 História **81**
A.2.2 Descrição **81**
A.2.3 Importação para o código **82**
A.2.4 Inicialização dos módulos **82**
A.2.5 Módulos **82**
A.2.6 Métodos **86**
Apêndice B - Código Fonte de Simulação Computacional – Algarismo Significativos, Ponto Material e Referencial **88**
Apêndice C - Código Fonte de Simulação Computacional – Encontro de móveis em movimento retilíneo e uniforme **92**
Apêndice D - Código Fonte de Simulação Computacional – Movimento retilíneo uniformemente variado **97**
Apêndice E - Código Fonte de Simulação Computacional – Lançamento Vertical **102**
Apêndice F – Código Fonte de Simulação Computacional – Lançamento Oblíquo **108**

Apresentação

Prezado(a) Professor(a)

Este trabalho é um guia metodológico para o estudo de Tópicos de Cinemática utilizando Modelagem Científica e a simulação computacional na forma de animação. A proposta consiste em transformar exercícios do livro didático em versões estendidas, que aprofundem os aspectos conceituais, e ensinar os alunos a desenvolverem questões animadas que evidenciem estes conceitos, utilizando o motor de jogos Pygame.

O conhecimento sobre o campo conceitual da Modelagem, será importante para orientar o aluno sobre as aproximações e idealizações do modelo físico e quais os referentes que serão considerados na simulação.

A Teoria que norteou a nossa proposta foi o Construcionismo, do professor Seymour Papert, educador que defende a ideia de que o aluno aprende melhor, quando imerso no seu "micromundo verdadeiramente interessante". A programação funciona como um "Lego", o aluno aprende a "encaixar as peças" e, com o tempo, será capaz de elaborar os códigos sozinho. Entretanto, os conceitos físicos que nortearão a animação computacional, não poderão ser redescobertos pelo aluno, precisarão ser ensinados pelo professor.

Organizamos a nossa explanação apresentando a fundamentação teórica para o ensino de Modelagem Científica no Ensino Médio e como abordá-la. Em seguida, apresentaremos o referencial teórico para o ensino de programação computacional e modelagem computacional. Apresentaremos a modelagem física, de 4 situações problemas, por meio de fichas de

atividades teóricas propostas, com questões que direcionam o aluno ao aprofundamento dos aspectos conceituais dos fenômenos físicos. Disponibilizamos, nos apêndices, um manual de Python básico e os códigos fontes de criação das animações utilizando o Pygame, em videoaulas.

Este material paradidático foi o resultado da pesquisa do Mestrado Nacional Profissional em Ensino de Física, do Polo 37- UFPA, finalizada em julho de 2022. Esperamos que ele possa motivar os seus alunos a gostarem da Física.

É importante, contudo, que o professor estude o que está sendo proposto, refaça, crie, e, posteriormente, aplique.

1. Fundamentação Teórica

1.1 Por que ensinar Modelagem Científica no Ensino Médio?

A modelagem consiste em representar um sistema, de forma simplificada e idealizada (BRANDÃO, ARAÚJO e VEIT, 2011, p. 508). No contexto científico, é importante por auxiliar em respostas que permitem compreender o mundo. No contexto educacional, os conceitos de modelagem científica, "imbricados" aos conteúdos de Física, permitem ao estudante "uma visão mais holística sobre a natureza e a construção do conhecimento científico" (BRANDÃO, ARAÚJO e VEIT, 2008, p. 10).

O excesso de formalismo apresentado nos conteúdos, provoca, na mente do aluno, um distanciamento entre os exercícios idealizados e o mundo real (BRANDÃO, ARAÚJO e VEIT, 2019, p. 86). É justamente esta dualidade de concepções, as que ele utiliza na sala de aula e as que utiliza no dia a dia, que lhe causam um desinteresse pela aprendizagem da disciplina, a crença de que estes conceitos não possuem uma utilidade prática no cotidiano (BRANDÃO, ARAÚJO e VEIT, 2008, p. 10-11).

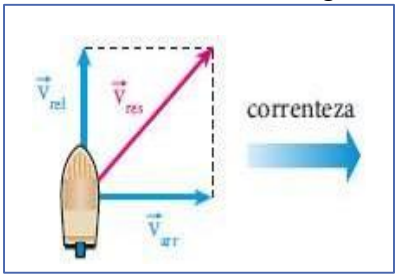
O processo de modelagem, no contexto científico, começa pela formulação de perguntas. Em seguida elaboram hipóteses e modelos conceituais. Para terem validade científica, estes modelos precisam ser confrontados com teorias gerais. É importante ainda destacar que estes modelos possuem um "domínio de validade", pois se concentram apenas nos aspectos de interesse. Isso significa, que em algum momento poderão falhar (BRANDÃO, ARAÚJO e VEIT, 2008, p. 11).

No contexto educacional, os conteúdos não são apresentados como modelos, mas como leis inquestionáveis (BRANDÃO, ARAÚJO e VEIT, 2019, p. 86). Não se discute o *domínio de validade* das equações apresentadas. (BRANDÃO, ARAÚJO e VEIT, 2008, p. 14). O aluno apenas é advertido de que a realidade é muito complexa, sem que lhe seja dado consciência de quão profundas são as limitações envolvidas nas situações-problema (BRANDÃO, ARAÚJO e VEIT, 2011, p. 532-533). Para exemplificar, tomemos a mudança de sinal no movimento retilíneo uniformemente variado. É possível imaginar um móvel, no mundo real, que se desloque na horizontal, submetido à uma aceleração constante realizando este tipo de manobra? É muito mais fácil imaginar um objeto jogado para cima submetido à aceleração da gravidade. Sendo assim este exemplo na horizontal é um pouco mais difícil de ser observado no cotidiano, por necessitar da ação de forças conservativas. Se o aluno tentar aplicar este raciocínio a um automóvel, em exercício escolar, fará os cálculos sem criticidade, caso não seja alertado para este detalhe.

Esta é a razão para os conceitos de modelagem científica estarem presentes no Ensino Médio, o fato de que contribui para "uma visão adequada à prática científica moderna, cuja essência está na criação de modelos" (BRANDÃO, ARAÚJO e VEIT, 2008, p. 11).

Ao iniciarmos a discussão de um problema, em sala de aula, sob o ponto de vista da modelagem científica, é preciso, inicialmente, estabelecer os referentes.

Figura 1 - Barco se desloca de uma margem à outra de um rio.



Fonte: HELOU, GUALTER e NEWTON, 2016, p. 74.

Considere um problema onde um barco atravessa um rio (Figura 1). Quais são os fatores que influenciam no tempo de travessia do barco e no seu deslocamento horizontal em relação às margens do rio? Estes devem ser os primeiros questionamentos. Talvez um aluno responda: "- O motor do barco". Neste caso, trazemos para o foco da discussão, de que seria a velocidade do barco. Outro fator é a velocidade da água devido à correnteza, que influencia no deslocamento horizontal do barco, tendo, por referência, as margens do rio. Devemos lembrar que os movimentos no sentido vertical e horizontal são independentes e cada um ocorre como se o outro não existisse (*Princípio da independência dos movimentos, de Galileu*). Neste caso, os referentes são: 1) o barco, 2) o trecho do rio de uma margem à outra, e 3) a água.

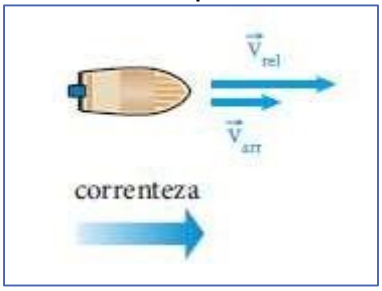
Feito isso, precisamos saber agora quais são as propriedades dos referentes que interessam na resolução do problema. O comprimento do barco não faz diferença neste caso, porque ele será tratado como ponto material, só interessa então a sua velocidade em relação às águas, ou velocidade relativa ($\vec{v}_{rel}$). Do trecho do rio, nos interessa a distância d de uma margem à outra. E das águas, nos interessa a sua velocidade em relação às margens do rio, chamada de velocidade de arrasto ($\vec{v}_{arr}$). Destes três elementos, 2 poderão ser colocados como parâmetros, por exemplo, a distância d e a velocidade de arrasto. Daí teremos como

variável, a velocidade relativa produzindo como resultados, a velocidade resultante, o tempo de travessia e ainda, o deslocamento horizontal do barco em relação às margens.

Observe que, quando tratamos o barco como ponto material, estamos fazendo uma simplificação, pois o comprimento do barco não influenciará no tempo da travessia, devido suas dimensões serem desprezíveis em relação à distância d. Sobre estas discussões, um aluno "ribeirinho", que já tenha vivenciado esta experiência, poderia fazer a seguinte observação: "- E o vento?" Aqui falamos sobre a idealização. O nosso modelo de estudo não considera forças dissipativas produzidas pelo ar. É preciso deixar claro para o aluno, que no mundo real, estes fatores estão presentes, para que ele não imagine existirem duas concepções sobre a realidade.

Considere agora que o barco desce o rio, como ilustra a Figura 2. Neste caso, o trecho do rio de uma margem à outra não interessa ao problema. Consequentemente, nossos referentes são apenas: 1) o barco e 2) as águas do rio.

Figura 2 - Barco desce o rio paralelamente às margens.



Fonte: HELOU, GUALTER e NEWTON, 2016, p. 74.

Para maior aprofundamento sobre o conteúdo de Modelagem Científica, indicamos o artigo (BRANDÃO, ARAÚJO e VEIT, 2008). O professor pode fazer implementações com o propósito de retransmissão aos alunos.

1.2 A importância da construção na aprendizagem

Como já falamos, a escola parte do abstrato para explicar o concreto, e supervaloriza esse formalismo. A construção é uma oportunidade de partir do concreto para o abstrato, é o movimento inverso. Um aplicativo, um circuito eletrônico, qualquer artefato que possa ser criado pelo aluno e possa ser analisado, criticado e "admirado, torna mais prazerosa a construção mental que ocorre na cabeça" (PAPERT, 2008, p. 137).

A construção é um "problema aberto", que pode apresentar várias soluções (OLIVEIRA, ARAUJO e VEIT, 2020). Neste sentido, o conteúdo de Física é um dos caminhos que o estudante deverá percorrer para solucionar o problema. A Física aqui não é o fim, mas o meio. É como estudar uma língua para cantar uma música, e após aprender a música, descobre várias expressões no idioma e figuras de pensamento que nem imaginava existirem, isso quando não aprende outros aspectos culturais do outro país. Por este motivo, PAPERT (2008, p. 136) afirma que "ensinar sobre X, não é a melhor maneira de melhorar o conhecimento de um estudante sobre o tópico X".

Na década de 1960, no ápice do movimento da Matemática Moderna, acreditava-se que o ensino do processo de pensamento científico era mais importante do que o conteúdo científico (PAPERT, 2008, p. 135). Este foi um equívoco epistemológico no sentido de dar autonomia aos estudantes, como se eles pudessem redescobrir as leis físicas, a despeito da ausência de conhecimento teórico que os cientistas possuem (GASPAR, 2004, p. 72-74). Esse era o resultado da "crença de um ensino de ciências fundado em abstrações" e "baseado na experimentação" (GASPAR, 2004, p. 74).

Segundo Piaget, a aprendizagem só é possível se o aluno tiver estrutura mental que possibilite a aprendizagem. Transferir a responsabilidade da aprendizagem para o aluno, significa também, responsabilizar a sua estrutura de pensamento, pelo balizamento do ensino. O cérebro humano funciona como um computador, que só pode processar as informações, cujos programas lógicos tenham sido instalados. Assim, a partir do trabalho de Piaget, a estrutura curricular foi modificada para buscar, não a estrutura lógica da Ciência, seu objeto, mas a estrutura lógica de pensamento do aluno, que é o destinatário do ensino (GASPAR, 2004, p. 79-80).

Uma situação hipotética, para ilustrarmos o que seria a estrutura mental necessária para se apropriar de um determinado conhecimento seria um pianista com anos de experiência que se predispõe a aprender um arranjo musical novo. A sua experiência em leitura de partituras e conhecimento sobre outros arranjos que praticou e executou durante anos, lhe possibilitará assimilar o novo arranjo em um intervalo de tempo muito curto, talvez algumas horas, enquanto um leigo, em Música, sem passar pelo mesmo processo, não poderia fazê-lo, nem com muito esforço, mesmo que levasse toda a sua vida. No primeiro caso, houve construção, e, no segundo, tentaríamos fazer a transmissão. Mas esta comparação inconcebível é muitas vezes imposta no Ensino de Física. Espera-se que o aluno interprete as equações e encontre resultados, sem que seja capaz de uma visualização mental do que estas equações representam.

O Construcionismo está entre estas teorias que promovem a discussão “transmissão versus construção”, com a diferença que se relaciona com a materialização de um produto. Papert visualiza o produto como um reforço positivo à construção mental. Uma das formas de fazer isso é por meio da programação computacional. A linguagem de programação é composta por

conjuntos, assim como peças de Lego, que permitem a construção de um programa (PAPERT, 2008, p. 137).

A importância da construção para a aprendizagem, é possibilitar mais aprendizagem com menos formalismo, menos instrução. Apresentar o concreto, e dele, partir para o abstrato. Neste caso, o aluno tem o reforço da visualização para compreender os modelos teóricos explicados no tópico anterior (PAPERT, 2008, p. 135).

1.3 O pensamento computacional

A linguagem Logo, criada em 1980, por Papert, possibilitou a entrada da programação na Educação. Entretanto, após o surgimento dos computadores pessoais, na maior parte do tempo, o uso dos computadores passou a se restringir no uso de "softwares de escritório", com editores de texto e planilhas, como se fossem máquinas de escrever modernas (VALENTE, 2016, p. 866). Em contrapartida, as tecnologias digitais de informação e comunicação (TDIC) têm provocado grandes transformações na sociedade, na cultura e no comércio (VALENTE, 2016, p. 866).

No intuito de capacitar os cidadãos, para serem criadores de tecnologias, ao invés de simples usuários, os EUA e vinte países da Europa substituíram a disciplina Informática, pelo tripé: Ciência da Computação, Tecnologia da Informação e Letramento Digital (VALENTE, 2016, p. 866).

Esta mudança de perspectiva é fruto do lançamento das bases do pensamento computacional, propostas por Jeannette Wing (VALENTE, 2016, p. 869). O artigo *"Computational Thinking It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use"*, Wing afirma que "pensar

como um cientista da computação requer vários níveis de abstração" (WING, 2006, p. 35). A professora defende que o pensamento computacional não deve ficar restrito a cientistas da computação, mas todos devem aprender. Inclusive, esta habilidade deveria ser acrescentada à capacidade analítica das crianças para melhorar a sua capacidade de análise da leitura, escrita e aritmética (WING, 2006, p. 33), ou seja, é aquele esforço de tornar a estrutura mental da criança preparada para lidar com conceitos novos. Papert, compartilha da mesma ideia quando defende que "ensinar a programar o computador e a pensar como desenvolver um projeto complexo" é como ensinar a pescar (PAPERT, 2008, p. 135).

Duas organizações, a International Society for Technology in Education[1] (ISTE) e a American Computer Science Teachers Association[2] (CSTA) trabalharam junto a pesquisadores da Ciência da Computação e da área de Humanas, para propor uma definição de pensamento computacional (VALENTE, 2016, p. 870).

Foram identificados nove conceitos:

COLETA DE DADOS
É a obtenção inicial de dados para processamento. A coleta de dados envolve identificar quais parâmetros e variáveis constituirão a modelagem do problema.

[1] Comunidade global de educadores que acreditam que a tecnologia pode solucionar problemas difíceis na Educação. Site oficial: https://www.iste.org/

[2] Sociedade que apoia e incentiva a Educação, no campo da Ciência da Computação, no Ensino Médio. Site oficial: https://www.csteachers.org/.

ANÁLISE DE DADOS

Consiste em identificar um algoritmo eficiente que resolva o problema. Isso pode ser feito a partir da análise das variáveis e parâmetros que constituem o problema. A análise pode ser de três tipos: viabilidade, correção e eficiência.

REPRESENTAÇÃO DE DADOS

A partir da análise de dados, é possível mostrar a relação entre as variáveis e parâmetros, em tabelas, gráficos e equações.

DECOMPOSIÇÃO DO PROBLEMA

Consiste em dividir um problema complexo em partes menores. Trabalhar com partes do problema, possibilita maior atenção à cada etapa e maior confiança para a compreensão do problema completo.

ABSTRAÇÃO

Permite ao aluno destinar o foco da análise de dados aos aspectos mais relevantes do problema, possibilitando análise crítica dos dados coletados e aproveitamento, do modelo utilizado, em outras aplicações.

ALGORITMO

Consiste em estabelecer os passos para a solução de um problema. Estes passos não precisam necessariamente corresponder a um problema de ordem matemática, mas pode se tratar de qualquer problema. O importante no algoritmo é estabelecer uma ordem, onde a entrada de dados que dependa da saída de algum processamento, esteja em sua

ordem correta.

AUTOMAÇÃO

É transferir para uma máquina, uma tarefa repetitiva que forneça a solução de um problema ou de suas partes. Para que o dispositivo execute a sequência de passos, além de uma linguagem que este dispositivo possa interpretar, é necessário que esta sequência tenha sido idealizada, como modelo, na mente humana que estabeleceu o algoritmo a ser seguido.

PARALELIZAÇÃO

Consiste em organizar tarefas que possam ser executadas de forma simultânea, ou seja, as partes não são interdependentes. Paralelizar partes da solução, otimiza o algoritmo.

SIMULAÇÃO

É a representação dinâmica de uma abstração, a última etapa. A simulação permite estudar os dados relevantes de um fenômeno qualquer que possa ser quantificado, por meio da análise gráfica ou outro tipo de representação visual, que indique o comportamento de variáveis em função das interferências externas ao sistema.

2. Criando animações com o Pygame

Disponibilizamos, no Apêndice A, uma síntese do que será utilizado, sobre a sintaxe da linguagem Python, na construção das simulações deste trabalho. Para facilitar a aprendizagem, disponibilizamos aulas gravadas, desde a instalação até a aplicação. O link <https://drive.google.com/drive/folders/1UrGnm0XZlbaVeiF9oTFtUdCfwdCHXeAg?usp=sharing>, deve ser colado no navegador. É fundamental que os alunos estudem esta parte, principalmente aqueles que nunca estudaram programação, para depois partirem para o estudo das animações com o Pygame.

O Pygame é um motor de jogos, que no inglês é conhecido como *game engine*. É um conjunto de bibliotecas, com funcionalidades que otimizam a criação de jogos (Figura 3). A nossa proposta é utilizar estas bibliotecas para otimizar a criação de animações. Estas animações serão questões animadas que os alunos aprenderão a criar para representar suas modelagens físicas.

Figura 3 - Exemplo de jogo criado com o Pygame.

Fonte: Disponível em http://renesd.blogspot.com/2018/03/windows-pypy-pygame-build-for-testing.html, acesso em 24 mar. 2022.

Abaixo apresentamos um exemplo. Disponibilizamos o código fonte no Apêndice B. Entretanto, para a sua execução será necessário estudar o conteúdo que apresentamos neste trabalho. A nossa sugestão é que, após estudar as animações propostas, o leitor tente criar este código. Depois compare com o código do apêndice.

Exemplo: Algarismo significativo e definição de ponto material.

Em (GASPAR, 2018, p. 29-33), encontramos a análise de um problema onde um trem atravessa uma ponte. No exercício em questão, o professor aborda o conceito de algarismos significativos, ponto material e referencial.

As observações apontadas pelo professor são:

- O que define se um corpo é ou não um ponto material, será a quantidade de algarismos significativos considerada, quando calculamos o intervalo de tempo da travessia, utilizando e não as dimensões do corpo. Dependendo do número de casas consideradas após a vírgula para diferenciar este intervalo de tempo, "nem uma formiga seria ponto material", esclarece.

- velocidade é um fator que influencia nesta classificação.

- O sinal de "aproximadamente igual", ~=, é uma redundância, pois toda igualdade, em Física, é uma aproximação.

Antes de criarmos a animação, ou seja, darmos movimento à questão, vamos inicialmente definir os referentes. O que determina o tempo da travessia é a velocidade do trem, o comprimento do trem e comprimento da ponte. Logo temos 2 referentes: trem e ponte. Os nossos dados de

entrada serão: comprimento do trem, comprimento da ponte e a velocidade do trem.

Após a criação do arquivo, será necessário importar o pygame para a aplicação e em seguida, inicializá-lo. Deverá ser definido o tamanho da janela de aplicação, o título e ainda, se necessário, a cor do fundo.

As imagens do trem e da ponte podem ser adquiridas em sites que disponibilizam grátis, ou se pode comprar a imagem e fazer download, como o https://craftpix.net/freebies/free-fairy-tale-game-backgrounds/ ou https://pixabay.com. Estas imagens podem ser encontradas na categoria *vectors,* no site. Antes de incluí-las no programa, é necessário verificar a quantidade de pixels que elas possuem, comparar com as dimensões da janela e no google digitar: dimensionar imagem. Daí colocar a quantidade de pixels referente à largura e altura, para ficar como figura padrão.

Na aplicação, deverá ser possível redimensionar a imagem do trem e da ponte, a partir da imagem padrão. Na tela de aplicação, devem ser apresentados textos com as informações: tamanho da ponte, tamanho do trem, posição do trem e o tempo. Estes recursos, o pygame disponibiliza por meios das funções `render` e `blit`.

O movimento do trem é criado por uma condicional `if`, que incrementa a sua posição da fração de velocidade equivalente à fração de tempo incrementada.

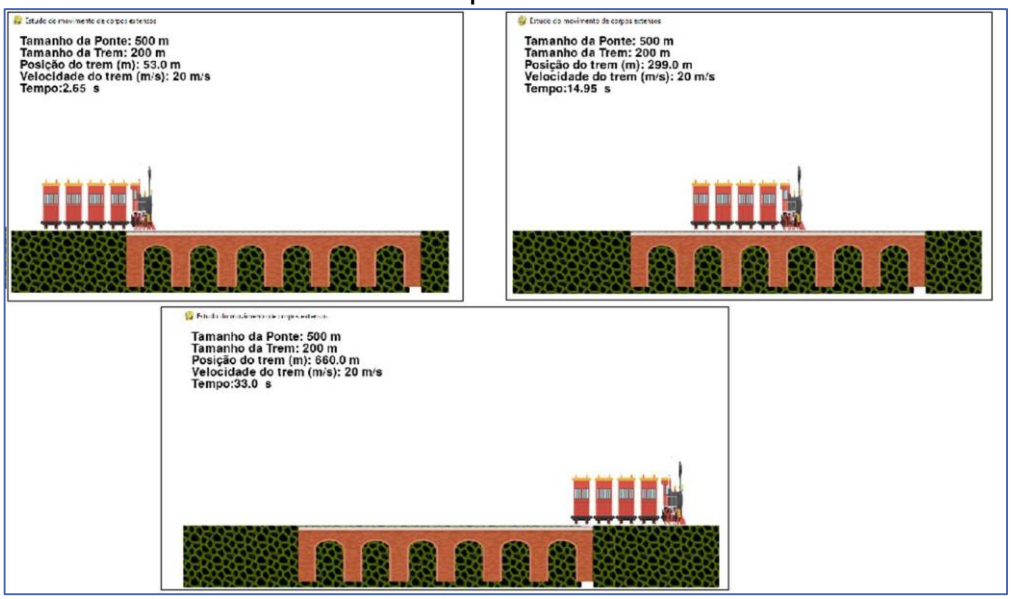
Visualmente, a execução iniciará com a entrada de dados (Figura 4).

Figura 4 - Entrada de dados da animação de um trem sobre uma ponte.

Fonte: Elaborado pelo autor (2022).

Na Figura 5, vemos a disposição do trem e da ponte em três instantes.

Figura 5 – Ilustração de três momentos da animação de um trem atravessando uma ponte.



Fonte: Compilação[3] do autor (2022).

[3] Montagem feita a partir das imagens coletadas no site Desenho Animado Vetores por Vecteezy.

3. Metodologia

Nossa sugestão de sequência didática, para aplicação desta metodologia, está disponível na tabela abaixo. As atividades de programação iniciam quando o professor julgar conveniente, ou seja, os vídeos podem ser fornecidos antes, durante ou após as aulas teóricas, desde que sigam a mesma sequência dos assuntos ministrados. Para facilitar, as atividades podem ser apresentadas em grupo:

Assunto	Atividade	Tempo
Introdução à modelagem científica	Aula expositiva sobre os conceitos de Modelagem	2 aulas de 50min
Introdução à linguagem Python	Acesso, pelos alunos às videoaulas de introdução à linguagem Python	10 dias
Estudo do movimento retilíneo e uniforme	Aula expositiva e atividade teórica	4 aulas de 50min
	Acesso, pelos alunos às videoaulas para criação de animação sobre encontro de móveis para reprodução do código	15 dias
Estudo do movimento retilíneo e	Aula expositiva e atividade teórica	4 aulas de 50min

uniformemente variado	Acesso, pelos alunos às videoaulas para criação de animação de MRUV para reprodução do código	15 dias
Estudo do lançamento vertical e queda livre	Aula expositiva e atividade teórica	2 aulas de 50min
	Criação da animação de lançamento vertical	10 dias
Estudo do lançamento oblíquo	Aula expositiva e atividade teórica	4 aulas de 50min
	Fornecimento do código fonte para o aluno lançar comentários	10 dias

3.1 Estudo do movimento retilíneo e uniforme

3.1.1 Resumo Teórico

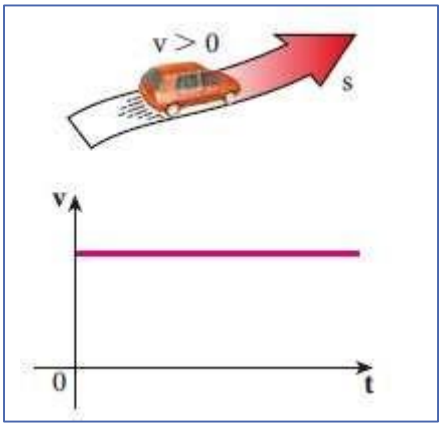
O estudo do movimento, em geral, deve iniciar pelo estabelecimento de um sistema de referência único (GASPAR, 2018, p. 34). A expressão algébrica de uma grandeza, que por natureza é vetorial, como a velocidade, é realizada pelo sinal (GASPAR, 2018, p. 35).

O movimento retilíneo, com velocidade constante, pode ser observado em alguns ambientes. Nas indústrias, é muito comum o uso de esteiras transportadoras, como pode ser conferido em https://www.youtube.com/watch?v=bPj5zbl9qwo. Os objetos sobre a

esteira estão com velocidade constante, até que o operário aperte o botão e suspenda o movimento de forma imediata.

Sobre este tipo de movimento, quando se trata de automóveis, caminhões e trens, quando equipados com piloto automático, é possível manter a velocidade constante por um longo tempo (HELOU, GUALTER e NEWTON, 2016, p. 34). Dependendo da velocidade que o automóvel se encontre, não é possível realizar uma interrupção brusca do movimento como as cestas na transportadora do item acima, a não ser que ele se choque com um obstáculo.

Figura 6 - Movimento progressivo.



Fonte: HELOU, GUALTER e NEWTON, 2016, p. 35.

Na Figura 6, o móvel se desloca no sentido positivo da orientação da trajetória, logo, sua velocidade terá o sinal positivo. O gráfico velocidade versus tempo é uma função constante positiva, indicando que a velocidade não muda com o tempo.

Quanto à posição, ela sofre um incremento de valores iguais, em intervalos de tempo iguais. Considere que a velocidade do móvel seja de 20m/s, no instante em que iniciou a contagem do tempo e se manteve constante em um intervalo de 5s, as posições estão na Tabela 1.

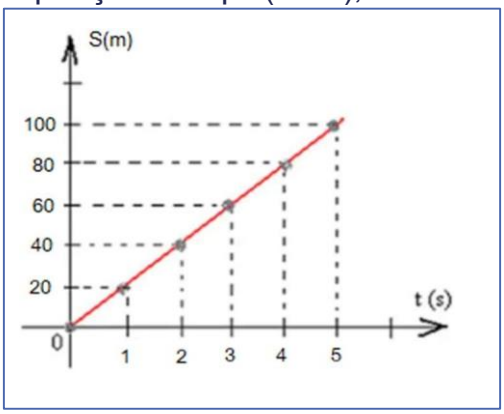
Tabela 1 - Posição em função do tempo (MRU) - movimento progressivo.

t (s)	0	1	2	3	4	5
s(m)	0	20	40	60	80	100

Fonte: Elaborado pelo autor (2022).

Logo, o gráfico será uma função crescente, dada na Figura 7.

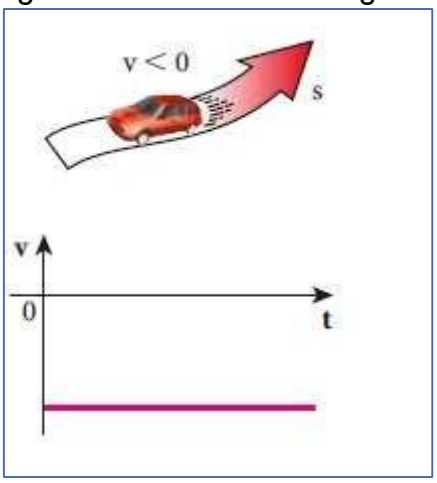
Figura 7 Gráfico posição x tempo (MRU), movimento progressivo.



Fonte: Elaborado pelo autor (2022).

No caso do móvel se deslocar no sentido contrário à trajetória:

Figura 8 - Movimento retrógrado.



Fonte: HELOU, GUALTER e NEWTON, 2016, p. 35.

Neste caso, o gráfico da velocidade em função do tempo, é uma função constante negativa. Considere então que o móvel esteja partindo

da posição 100m, e anda 20m a cada segundo, contra a orientação da trajetória.

Tabela 2 Posição em função do tempo (MRU) - movimento retrógrado.

t (s)	0	1	2	3	4	5
s(m)	100	80	60	40	20	0

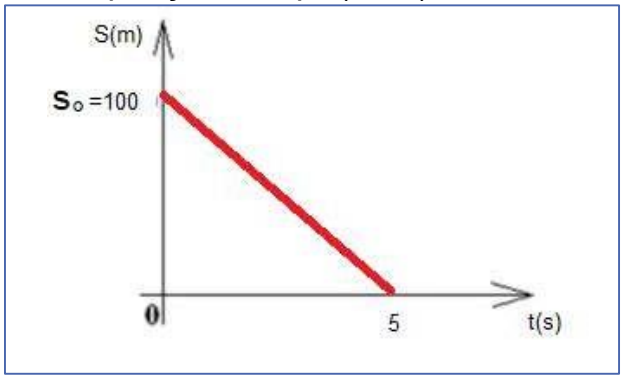
Fonte: Elaborado pelo autor (2022).

A posição do móvel é dada pela equação:

$$s = s_0 + vt$$

onde s_0 é a posição inicial do móvel e v é a velocidade constante que caracteriza o movimento uniforme.

Figura 9 - Gráfico posição x tempo (MRU), movimento retrógrado.



Fonte: Elaborado pelo autor (2022).

3.1.2 Atividade teórica proposta

P.1 - Encontro de dois móveis

A questão apresentada abaixo estabelece um intervalo de velocidades e distâncias entre os móveis. O professor deve levar o aluno a raciocinar sobre os limites de velocidade estabelecidos nas vias de sua

cidade. Para isso terá que fazer a conversão de m/s para km/h. Oriente-os a identificar os referentes do problema (os dois carros), e quais grandezas destes referentes são necessárias para resolver o problema (VA, VB, SA, SB). Mostre aos alunos que o referencial é único para os dois móveis. A partir das equações que definem os movimentos, solicite para que os alunos discutam qual a condição para que ocorra o encontro.

Questão 1. Suponha que você observe a seguinte cena: em uma via retilínea de duas pistas, como ilustra a figura abaixo, estejam trafegando dois carros em sentidos opostos, um em cada pista, distando em linha reta uma distância X um do outro, de modo que você observa os movimentos dos carros até o instante em que os carros se cruzam na via. Suponha ainda que os carros se deslocam com velocidades constantes, as quais possam assumir em módulo qualquer valor dentro do intervalo [10 m/s, 25 m/s], sendo a distância X dada no intervalo [200m, 600m].

Figura 10 - Encontro de dois carros em uma via.

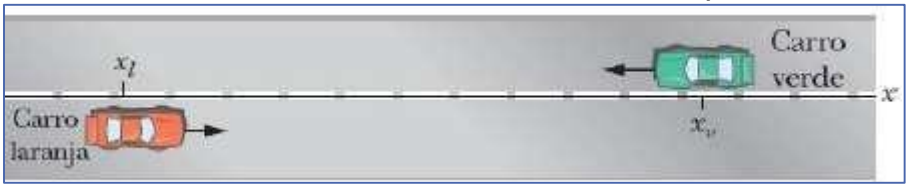
Fonte: Adaptação[4] feita pelo autor (2022).

Escolha valores para as velocidades dos carros e para a distância X, e resolva os questionamentos a seguir:

[4] Figura original disponível em https://pxhere.com/pt/photo/1097533.

a) Faça uma representação cinemática para a situação de cruzamento de dois carros através de um desenho, constando de: 1. Defina um eixo de coordenadas para a análise do movimento dos carros; 2. Identifique as grandezas cinemáticas dadas inicialmente, através da escolha apropriada de valores numéricos. Sugerimos, por exemplo, um desenho como o apresentado na Figura 11, o qual você pode utilizar.

Figura 11 – Dois carros se movimentando em sentidos opostos em uma via.



Fonte: HALLIDAY, RESNICK e WALKER, 2016.

b) Determine as equações de movimento para cada automóvel, justificando fisicamente cada equação, analisando a partir da teoria física que você acha adequada para analisar a situação.

c) A partir do início da observação do movimento dos carros, determine o instante de tempo e a posição em que os carros se cruzam na via. Explique fisicamente como resolver.

d) Dada as equações de movimento dos carros, determine agora graficamente o instante de tempo e a posição do cruzamento dos carros.

3.1.3 Atividade de modelagem computacional

a) Propomos agora que você realize a análise desta situação de cruzamento de dois carros, realizando a animação computacional. Para isso você deverá assistir aos vídeos disponibilizados sobre a programação, em linguagem Python, disponibilizados no Google Drive, de um modelo computacional que simula o encontro de móveis, executando Movimento Retilíneo e Uniforme: MU AULA1, MU AULA2, MU AULA3, MU AULA4 e MU AULA5. Você deve estudar a animação reescrevendo o código, na IDE Pycharm. Em seguida, implemente uma modificação neste programa que possibilite escolher as posições iniciais dos móveis. Após esta modificação, realize 5 entradas de posição inicial e velocidade para os móveis, e gere um print no gráfico posição versus tempo, de cada entrada, e copie no espaço abaixo, escrevendo ao lado o par de equações dos móveis correspondentes a cada entrada.

   Link da aula:
   (https://drive.google.com/drive/folders/1in4IC2YJuo5hMYXwji00hTHTfCu29vSf?usp=sharing)

Comentários sobre a aplicação e código fonte:

O código fonte foi construído obedecendo à seguinte sequência:

1- Entrada dos dados.
2- Leitura das variáveis.
3- Impressão das imagens e textos na tela a partir da leitura das variáveis.
4- Incremento das variáveis enquanto a condição for atendida.
5- Retorno ao passo 2, caso o passo anterior tenha sido atendido.

Espera-se que ao final da atividade, o aluno seja capaz de:

- Identificar o referencial da trajetória estabelecida no problema, a partir dos gráficos de posição em função do tempo.
- registrar as funções horárias das posições, a partir dos gráficos gerados.

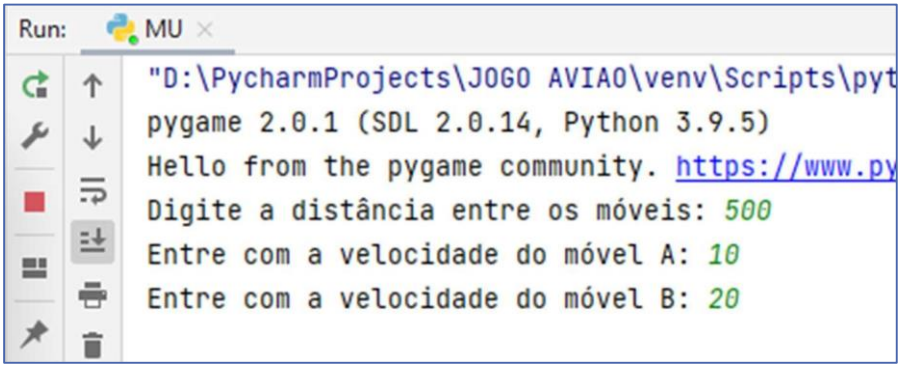
O código que simula a animação está disponível no Apêndice C.

A modelagem física do problema inicia estabelecendo-se os referentes: carro A e carro B.

No exemplo, colocamos o carro A na origem. A posição do carro B, será a distância entre os móveis. Além do movimento dos móveis, a animação deverá gerar o gráfico da posição em função do tempo (Sxt) dos dois móveis no mesmo eixo de coordenadas, com o objetivo de evidenciar a posição do encontro. A entrada de dados fornecida pelo usuário será constituída de: velocidade do carro A, velocidade do carro B e distância entre os móveis.

Estas informações serão inseridas pela função `input`. A Figura 12, mostra a entrada de dados.

Figura 12 - Entrada de dados associada aos movimentos dos dois carros.



Fonte: Elaborado pelo autor (2022).

Dentro do `while`, as posições dos móveis são incrementadas da fração de suas velocidades no intervalo do *loop*, produzindo o efeito de movimento dos carrinhos.

```
    if SB > 0:
            SA = SA + VA/20
            SB = SB - VB/20
            tempo = tempo+0.05
```

Esta equação é $S= S_0 + Vt$, onde t = 0,05s, que é igual $\frac{1}{20}$ s.

A apresentação dos carrinhos na tela, é feita no seguinte trecho do código:

```
    janela.blit(carrinhoA, (int(SA),250))
    janela.blit(carrinhoB, (int(SB),320))
```

Para armazenamento das posições dos móveis e do instante de cada posição, utilizamos a estrutura lista:

```
        ordenadaA = []
        ordenadaB = []
        abcissa = []
                  .
                  .
                  .
        ordenadaA.append(int(SA))
        ordenadaB.append(int(SB))
        abcissa.append(20*tempo)
```

Utilizamos a condicional `if` para determinarmos o instante do encontro:

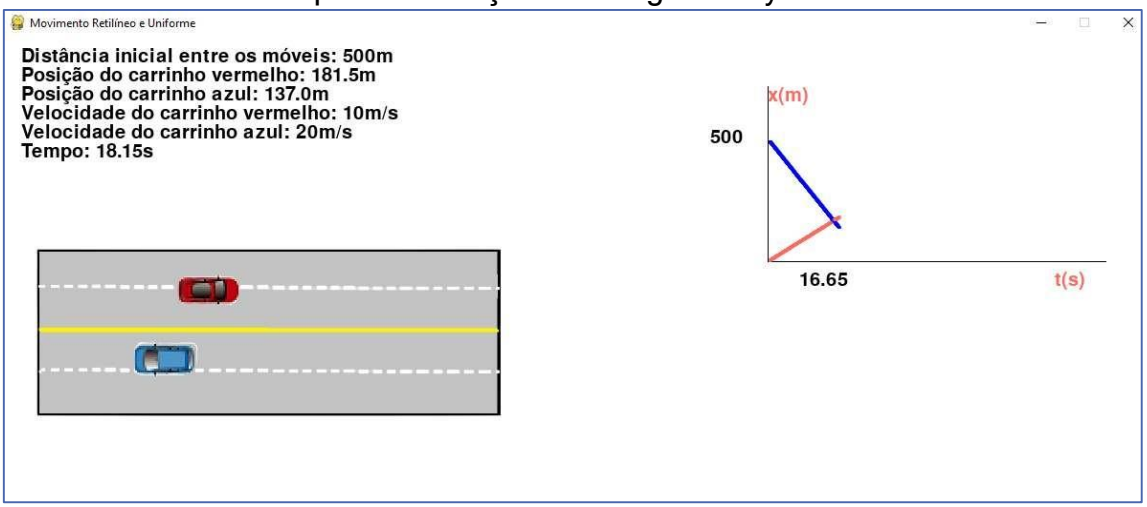
```
if SB + VB/20> SA > SB - VB/20:

    temp_enc=font.render(str(round(tempo,2)),1,(0,0,0))
    posicaox = 800+2*round(tempo,2)
    janela.blit(temp_enc, (posicaox, 120+ SB0/4))
    flag = 1
```

Em programação, não podemos colocar a igualdade `SA = SB` na condicional, pois, diferente do que acontece na modelagem matemática, o incremento aqui é discreto e não contínuo. Os valores da igualdade podem nunca acontecerem.

A variável `flag` foi criada com o valor zero, para indicar que o encontro entre os móveis ainda não ocorreu. Ao se tornar verdadeira a condição `if`, a `flag` deve mudar o valor para 1. Observamos no código que, a posição de `SA` deverá estar em um intervalo de `VB/20`, antes ou depois de `SB`, para que a condição `if` seja verdadeira. O comando `janela.blit` imprime o instante do encontro no gráfico, como pode ser observado na Figura 13.

Figura 13 - Visualização da animação computacional do encontro de dois automóveis, após a execução do código em Python.



Fonte: Elaborado pelo autor (2022).

A apresentação do gráfico é feita pelos comandos:

```
    ponto_azul = font.render(".", 10, (0,0,255))
    ponto_verm = font.render(".", 10, (255, 105, 97))
    text_sm = font.render("x(m)", 5, (255, 105, 97))
    text_ts = font.render("t(s)", 5, (255, 105, 97))

    pygame.draw.line(janela, (0,0,0), (800,51), (800, 110+
    SB0/4) )
    pygame.draw.line(janela, (0, 0, 0), (800, 110+SB0/4),
(1200, 110 + SB0/4))

    janela.blit(text_sm, (800, 51))
    janela.blit(text_ts, (1100, 120+SB0/4))

    ordenadaA.append(intSA))
    ordenadaB.append(intSB))
    abcissa.append(20*tempo)

        i=0
```

```
while i<len(ordenadaA):
    janela.blit(ponto_verm, (800 +abcissa[i]/5, 95+SB0/4-
                        ordenadaA[i]/4))
    janela.blit(ponto_azul, (800 + abcissa[i]/5, 95+SB0/4
                       - ordenadaB[i]/4))

                                i += 1
```

Inicialmente, são armazenados os pontos azul e vermelho nas variáveis `ponto_azul` e `ponto_verm` e os textos "x(m)" e "t(s)" nas variáveis `text_sm` e `text_ts`, respectivamente.

Depois, foi utilizado o `pygame.draw.line` para desenho dos eixos do gráfico. E, por fim, o laço de repetição `while`, a cada loop, lerá o tamanho da lista `ordenadaA` e imprimirá os pontos na tela de acordo com este tamanho.

3.2 Estudo do movimento retilíneo uniformemente variado.

3.2.1 Resumo Teórico

No movimento retilíneo uniformemente variado, a velocidade sofre incremento de um valor fixo na unidade de tempo. Este incremento é chamado de aceleração.

Considere um móvel que se desloque a partir do repouso ($V_0 = 0$) e tem sua velocidade incrementada em 2 m/s a cada segundo.

Tabela 3 - Velocidade em função do tempo (MRUV) - movimento acelerado.

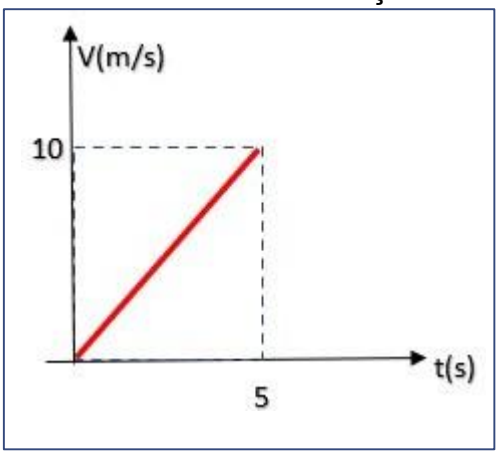
t (s)	0	1	2	3	4	5

v(m/s)	0	2	4	6	8	10

Fonte: Elaborado pelo autor (2022).

O gráfico abaixo indica velocidade em função do tempo:

Figura 14 - Gráfico da velocidade em função do tempo (MRUV).



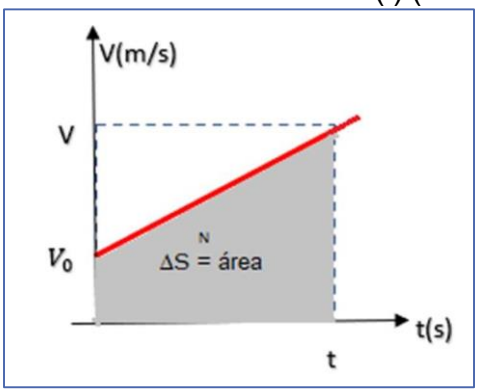
Fonte: Elaborado pelo autor (2022).

Neste caso, a velocidade em um instante qualquer, é dada por:

$$v = v_0 + a.t$$

Para determinar a função horário da posição, consideremos que o móvel parte de uma velocidade v_0 qualquer e atinja uma velocidade v (Figura 15) .

Figura 15 - Área sob a curva v(t) (MRUV).



Fonte: Elaborado pelo autor (2022).

A área sob a curva, é numericamente igual ao deslocamento do móvel. Se considerarmos a área do trapézio acima, concluímos que:

$$\Delta S = V_o t + \frac{a}{2} t^2$$

Partindo da definição de $\Delta S = S - S_0$:

$$S = S_0 + V_o t + \frac{a}{2} t^2$$

O movimento uniformemente possui quatro classificações:

Tabela 4 Tabela de classificação dos movimentos no MRUV.

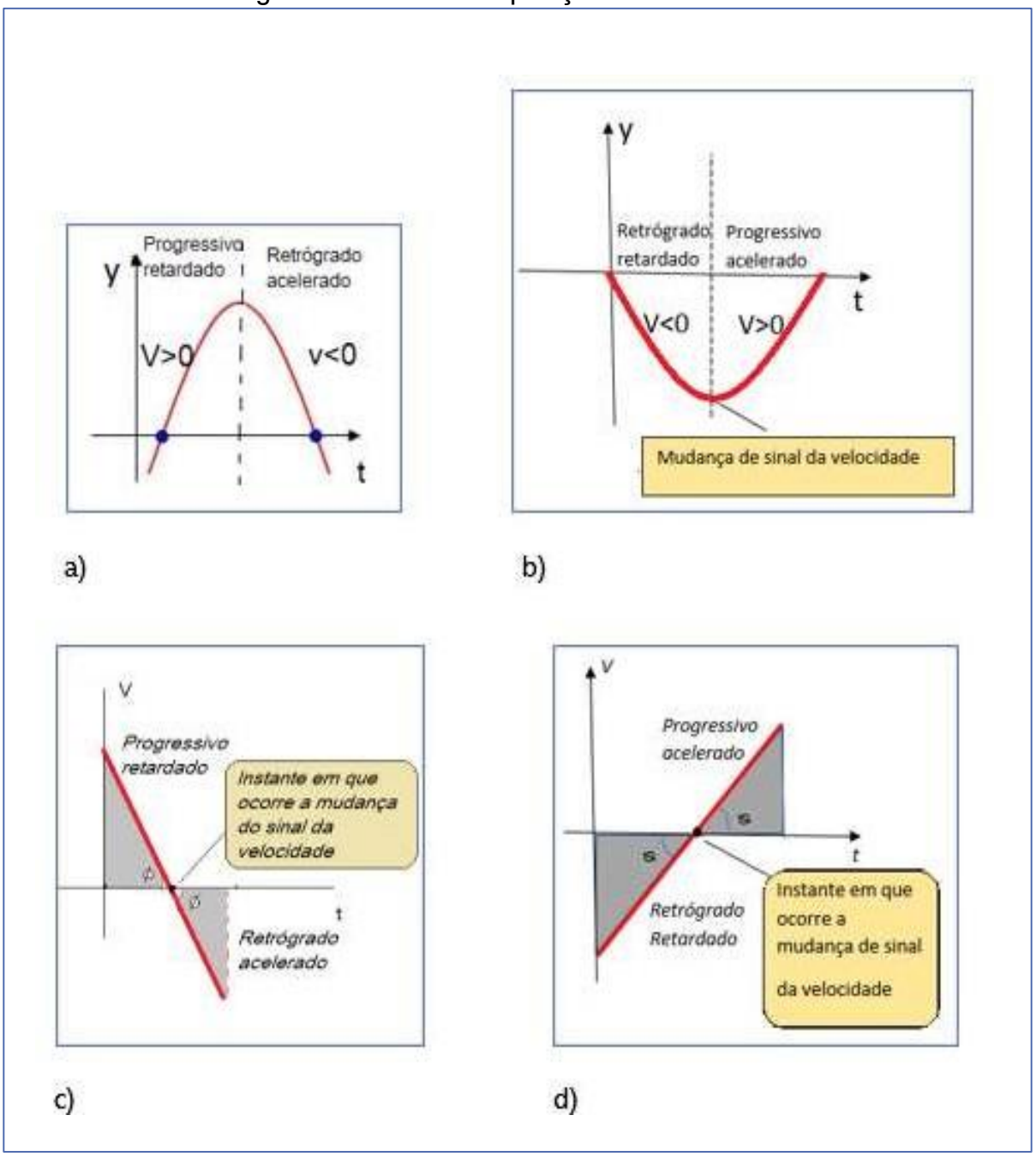
Classificação	Descrição	Sinais das grandezas
Progressivo acelerado	O módulo da velocidade do móvel aumenta, se deslocando a favor da trajetória.	V>0 a>0
Retrógrado acelerado	O módulo da velocidade do móvel aumenta, se deslocando contrário à orientação da trajetória.	V<0 a<0
Progressivo retardado	O módulo da velocidade do móvel diminui, se deslocando a favor da orientação da trajetória.	V>0 a<0
Retrógrado retardado:	O módulo da velocidade do móvel diminui, se deslocando contrário à orientação da trajetória.	V<0 a>0

Fonte: Elaborado pelo autor (2022).

No modelo do MRUV, o movimento retardado atinge seu ápice quando a velocidade do móvel chega a 0. Daí ocorre a mudança de sentido, ou mudança de sinal da velocidade.

No lançamento vertical, o móvel está submetido à aceleração g da gravidade. Orientando-se a trajetória para cima, na subida do lançamento, o movimento é progressivo retardado e na descida, retrógrado acelerado (Figura 16.a e Figura 16.c). Se invertermos a orientação, o movimento se torna retrógrado retardado na subida e progressivo acelerado na descida (Figura 16.b e Figura 16.d).

Figura 16 - Gráficos de posição e velocidade no MRUV.



Fonte: Elaborado pelo autor (2022).

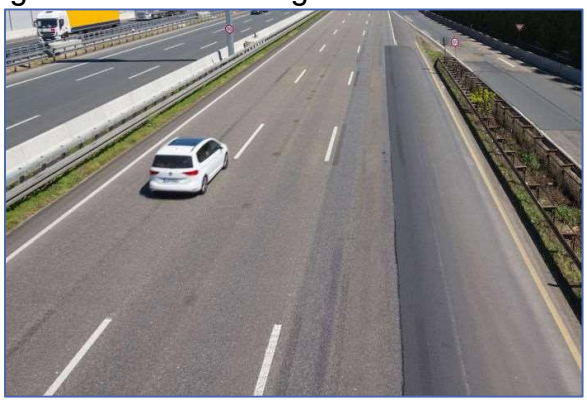
3.2.2 Atividade teórica proposta

P.2 Carro trafegando em via retilínea

A questão propõe que o movimento retilíneo uniformemente variado ocorre em um determinado trecho, no qual o motorista pressiona o freio bruscamente. O professor deve questionar os alunos sobre a mudança de sinal da velocidade que não é observada neste fenômeno. Ao chegar à velocidade zero, o móvel deverá parar.

Questão 2. Suponha que você observe um carro trafegando em uma via retilínea, como ilustra a figura abaixo, sendo que a velocidade do carro possa aumentar ou reduzir, conforme o motorista alterne entre o acelerador e o freio.

Figura 17 - Carro trafegando em via retilínea.



Fonte: Ver nota de rodapé[5].

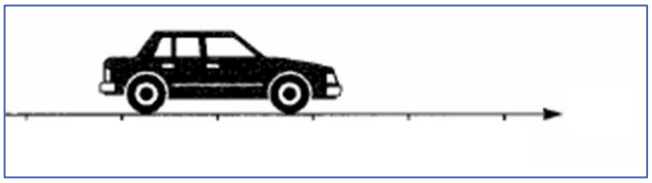
Considere ainda que o automóvel esteja com velocidade de 72 km/h, no início da observação, e que após o tempo de 2 s, a velocidade observada para o automóvel seja de 36 km/h. Provavelmente você concorda que o movimento do carro seja bem simples, já que apresenta poucos parâmetros, entretanto, com conhecimentos básicos de Cinemática é

[5] Disponível em https://brasilescola.uol.com.br, acesso em 2 out. 2021

possível analisarmos o movimento do carro e obtermos reflexões interessantes! Vejamos então os seguintes questionamentos.

(a) Faça a representação cinemática para esta situação do movimento do carro através de um desenho, constando de: 1. Defina um eixo de coordenadas para a análise do movimento; 2. Identifique no desenho as grandezas cinemáticas dadas inicialmente, conforme o enunciado. Sugerimos, por exemplo, um desenho como o apresentado abaixo, o qual você pode utilizar.

Figura 18 – Movimento de um carro mostrando o eixo de coordenadas.



Fonte: NUSSENZVEIG, 2002, p. 23.

(b) Assumindo que o carro se desloque com aceleração constante, determine o valor desta aceleração, e identifique a orientação do vetor aceleração no eixo de coordenadas do desenho.

(c) Construa o gráfico para a velocidade do carro versus o tempo, desde o instante inicial t = 0 s, até t = 8 s e classifique o movimento do carro. Para isto, observe o que está acontecendo com os valores da velocidade em relação a aceleração e o eixo de coordenadas do movimento. Também, a partir da análise do movimento dada pelo gráfico, você acha possível este comportamento na realidade?

3.2.3 Atividade de modelagem computacional

(a) Propomos agora que você realize a animação do movimento em linha reta do carro, que adquire aceleração constante enquanto a tecla direcional do teclado é pressionada, ou seja, ele executa um MRUV, sendo possível observar também o gráfico iterativo da velocidade com tempo, à medida que o carro se move. Para isso, você deverá assistir aos vídeos disponibilizados sobre a programação, em linguagem Python, disponibilizados no Google Drive, de um modelo computacional de um automóvel executando o MRUV, que apresenta o gráfico iterativo da posição com tempo. Os vídeos são: MUV AULA1, MUV AULA2, MUV AULA3, MUV AULA4. Você deve estudar a animação, reescrevendo o código na IDE Pycharm e implementando o gráfico velocidade por tempo. No entanto, por questão de escala, adote o valor $1m/s^2$ ou -1 m/s^2 para as acelerações. Faça *prints* da sua animação e de suas dúvidas e quando sua animação estiver plenamente funcionando, grave um vídeo comentando o que está acontecendo com o movimento do carro, e disponibilize o acesso para o professor.

(b) Analisando o resultado da animação do carro associado ao gráfico de sua velocidade, como você contrapõe com o que foi analisado ao final da letra (c) da atividade teórica, ou seja, o modelo da animação e a observação de cenas reais do movimento de um carro?

Link da aula:

https://drive.google.com/drive/folders/1JUksB4TEZbI0OOU5Q7TZSotZCovSBefJ?usp=sharing

Comentários sobre a aplicação e código fonte:

Este modelo foi adaptado de GASPAR (2018, p. 38) e foi representado pela animação de um automóvel que recebe uma aceleração de módulo 1 m/s², no sentido da tecla direcional em que for solicitado. Para simplificação do modelo, se a tecla direcional permanecer pressionada no instante em que a velocidade chegar a zero, o carro mudará o sentido do movimento. É compreensível que, no mundo real, um automóvel não faz esse tipo de manobra, sendo esta mais apropriada ao lançamento vertical. Entretanto, o objetivo desta aplicação é justamente levar o aluno a compreender as limitações do modelo físico e sua inadequação para explicar a mudança de sentido de um automóvel.

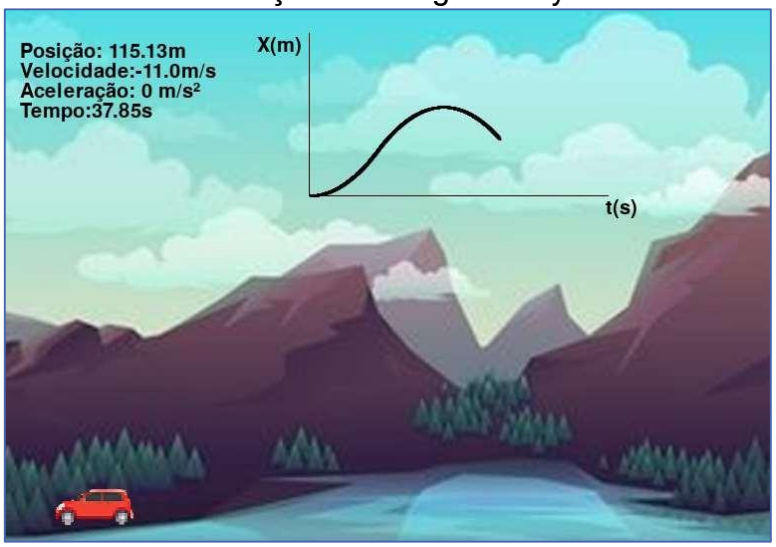
A Figura 19, dada a seguir, mostra o resultado da animação computacional para o carro em MRUV. O estudo da execução desta aplicação, deverá mostra os seguintes aspectos qualitativos:

- O sinal das grandezas vetoriais só depende do referencial.
- A aceleração negativa não é sinônimo de freamento ou desaceleração.
- Em um MRUV, se a aceleração se mantém constante, o móvel quando para, não fica parado, continua em movimento no sentido oposto ao anterior à parada.' (GASPAR, 2018, p. 38)

E ainda acrescentamos:

- Se a aceleração do móvel está no sentido contrário ao movimento, o módulo da velocidade diminui e o movimento é retardado.
- Quando o movimento é acelerado progressivo, a concavidade do gráfico da posição versus o tempo é aberta para cima e se encontra no lado crescente.
- Quando o movimento é retardado progressivo, a concavidade do gráfico da posição versus o tempo é aberta para baixo e se encontra no lado crescente.
- Quando o movimento é acelerado retrógrado, o gráfico da posição versus o tempo é uma concavidade aberta para baixo e se encontra no lado decrescente.
- Quando o movimento é retardado retrógrado, o gráfico da posição versus o tempo é uma concavidade aberta para cima e se encontra no lado decrescente.

Figura 19 – Visualização da animação computacional de um automóvel em MRUV, após a execução do código em Python.



Fonte: Compilação[6] do autor (2022).

[6] Montagem a partir das imagens coletadas no site https://craftpix.net/freebies/free-fairy-tale-game-backgrounds/ e https://br.pinterest.com/pin/288934132347483921/

O código fonte completo, referente à execução desta animação, está disponível no Apêndice D.

O controle das teclas é apresentado no seguinte trecho do código:

```
comandos = pygame.key.get_pressed()

if comandos[pygame.K_RIGHT] and x>=-190:
   velocidade = velocidade + 0.05    #v = vo + a (delta t)
   text_a = font.render("Aceleração: 1 m/s²", 1, (0,0,0))


   if velocidade >=0:
       carro= pygame.image.load('carrinho_direita.png')


if comandos[pygame.K_LEFT] and x <=80:
    velocidade = velocidade - 0.05
    text_a = font.render("Aceleração: -1 m/s²", 1, (0, 0, 0))

    if velocidade <= 0:
        carro = pygame.image.load('carrinho_esquerda.png')
```

A variável `comandos` está armazenando a tecla pressionada pelo usuário, por meio do método `get_pressed()` que está associada ao módulo `key`.

Neste exemplo, o sentido da trajetória está orientado para a direita. Logo, se o carrinho está se deslocando para a direita, o movimento é progressivo e a velocidade é positiva. Se o carrinho está se deslocando para a esquerda, o movimento é retrógrado e a velocidade é negativa.

A condicional `if` estabelece que, se a tecla direcional direita for pressionada e o x da imagem na variável `fundo` (porque é o fundo que se desloca e não o carro) estiver acima do valor -190, a velocidade de

deslocamento do fundo para esquerda deve ser incrementada de 0.05 pixels. O efeito visual produzido é de o automóvel sofrer uma aceleração para a direita. Se isso acontece enquanto o carrinho está se deslocando para a direita, ele sofre uma aceleração, pois o módulo da velocidade aumenta. Se ocorre quando o carrinho está se deslocando para a esquerda, a sua velocidade é decrementada de 0.05 pixels produzindo o efeito de que o carrinho desacelera, pois o módulo da velocidade diminui.

Analogamente, quando a tecla direcional esquerda é pressionada e a posição x da imagem do fundo é menor ou igual a 80, o carrinho sofre uma aceleração para a esquerda. Sendo assim, se ele se movimenta para a direita, sofre uma desaceleração, pois o módulo da velocidade diminui. Se ele se desloca para a esquerda, sofre uma aceleração, pois o módulo da velocidade aumenta.

Estes limites de -190 e 80 foram estabelecidos em função das dimensões em pixels da imagem do fundo.

O pensamento computacional atua para organizar o fenômeno aceleração das quatro situações em apenas duas condicionais `if`.

O incremento da velocidade é de 0.05 e -0.05, pois a variável `velocidade` é justamente a função horária da velocidade do MRUV, sendo que definimos o módulo da aceleração como 1m/s² e o tempo está sendo incrementado de 0.05 a cada loop.

```
tempo = tempo+0.05
```

O modelo físico despreza o atrito. Sendo assim, ao se soltar a tecla direcional, o móvel deve manter a velocidade constante, conforme a segunda linha do trecho do código abaixo. Neste trecho é mostrado ainda,

um tratamento de bordas para que o carrinho não fique travado nas extremidades:

```
if x>-190 and x<80:
     x-=velocidade*0.05

  else:

    if x>80:

      velocidade = 0
      x=78


    if x<-190:
      print("agora sim" + str(x))
      velocidade = 0
      x =-188
```

Além do movimento do carro, a animação deve apresentar um gráfico da posição em função do tempo.

Para armazenar a posição de carro e o respectivo tempo, são criadas as listas:

```
ordenada=[]
abcissa=[]
```

A construção do gráfico da posição em função do tempo, está disponível nos seguintes trechos:

```
text_sm = font.render("X(m)", 1, (0, 0, 0))
text_ts = font.render("t(s)", 1, (0, 0, 0))
ponto = font.render(".",5,(0, 0, 0))

ordenada.append(int(x)/2)
abcissa.append(5*tempo)
```

```
janela.blit(text_sm, (258, 54))
janela.blit(text_ts, (608, 217))

pygame.draw.line(janela, (0, 0, 0), (310, 51), (310, 214))
pygame.draw.line(janela, (0, 0, 0), (310, 214), (610, 214))


i=1

while  i<len(ordenada):

    janela.blit(ponto,(310+abcissa[i], 200 +ordenada[i]))
    i += 1

    if len(ordenada)==1200:
          del ordenada[1]
```

O fato de deletar o primeiro elemento da lista, quando o número de elementos chega a um determinado valor, permite que no loop seguinte, o elemento de posição x+1, se torne o elemento de posição x, deslocando o gráfico para a esquerda.

3.3 Estudo do lançamento vertical e queda livre

3.3.1 Resumo Teórico

O modelo de lançamento vertical estudado em Cinemática utiliza como referenciais, além do objeto que é lançado na vertical, a Terra. Embora o estudo da Cinemática não faça referência à força peso, sabemos que a aceleração g é resultado da atração gravitacional.

Do objeto lançado, interessa para a solução do problema, a posição do lançamento, em relação a um eixo de coordenadas, a sua velocidade inicial e o ângulo de lançamento. Da Terra, nos interessa a aceleração g com que ela atrai os corpos, no local do lançamento.

Considerando alturas muito pequenas, podemos desconsiderar a resistência do ar, idealização. Podemos ainda considerar a aceleração da gravidade sem casas decimais, arredondando para 10m/s², o que seria a aproximação.

O gráfico de subida e descida, foi discutido em 3.2.1, pois se trata de um MRUV. A discussão aqui se resume a dois pontos: o tempo de subida e a altura máxima.

Considere um objeto lançado para cima com velocidade inicial v_0. No instante em que o objeto alcançar a altura máxima, a sua velocidade será nula. Aplicando a função horária da velocidade para o MRUV, temos:

$$v = v_0 - gt$$

$$0 = v_0 - gt$$

$$t = \frac{v_0}{g}$$

Onde $t = \frac{v_0}{g}$, é o tempo de subida.

Outra maneira de enxergar é que, na altura máxima, o módulo do vetor $\vec{g}t$, se iguala ao módulo do vetor $\vec{v}_0$.

A altura máxima atingida, substituindo o tempo de subida na função horária da posição.

$$s = s_0 + v_0 t - \frac{g}{2} t^2$$

$$s = s_0 + v_0 \frac{v_0}{g} - \frac{g}{2}\left(\frac{v_0}{g}\right)^2$$

$$s = s_0 + \frac{v_0^2}{g} - \frac{(v_0)^2}{2g}$$

$$s - s_0 = \frac{(v_0)^2}{2g}$$

Sendo $s - s_0$ a altura máxima ($H_{máx}$):

$$H_{máx} = \frac{(v_0)^2}{2g}$$

A queda livre seria a segunda parte do lançamento vertical. O objeto é abandonado de determinada altura H e, desprezando-se a resistência do ar, cai submetido apenas à aceleração da gravidade.

O tempo de queda para atingir o solo, é dado por:

$$H = \frac{g}{2}t^2$$

$$t_q = \sqrt{\frac{2H}{g}}$$

3.3.2 Atividade teórica

P.3 Objeto lançado para cima

A questão apresenta um modelo de lançamento vertical, onde a resistência do ar é desprezada (idealização). Os referentes são o objeto e a Terra. Mostre ao aluno que o referencial estabelecido não muda de sentido quando o objeto passa pelo ponto mais alto. Isso é importante para o aluno compreender que, se estiver orientando a trajetória para cima, o sinal da aceleração da gravidade será sempre negativo. Chame a atenção para o fato de que o modelo é idealizado, pois despreza a resistência do ar. A partir deste ponto, esperamos que o aluno tente

desenvolver o código fonte, com auxílio do professor. No Apêndice E, é fornecido um código fonte de exemplo para o lançamento vertical, o qual o professor poderá estudar e orientar os alunos no desenvolvimento de suas simulações.

Questão 3. Suponha que você observe um menino lançando uma bola para o alto, verticalmente (Figura 20), e que após alguns segundos, a bola, ao invés de retornar para as suas mãos, cai direto no solo. A bola é lançada com velocidade inicial de 10m/s a partir de 2,0m acima do solo. Considere a aceleração da gravidade no local igual a 10m/s². Despreze a resistência do ar. Considere $\sqrt{5}$=2,24.

Figura 20 Menino jogando uma bola para o alto.

Fonte: Ver nota de rodapé[7].

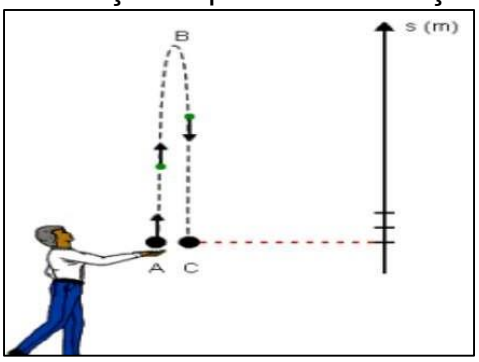
A partir desta observação, podemos fazer algumas interpretações sobre os resultados encontrados nas equações e outras observações interessantes. Para isso, faça o que se pede abaixo:

(a) Faça a representação cinemática para esta situação do lançamento através de um desenho, constando de: 1. Defina um eixo de coordenadas para a análise do movimento; 2. Identifique no desenho as grandezas cinemáticas dadas inicialmente, conforme o enunciado. Indique também o sentido das grandezas no início do movimento e no final. Represente v=0 no ponto em que este valor é

[7] Disponível em https://mundoeducacao.uol.com.br/fisica/lancamento-vertical-para-cima.htm, acesso em 29 mar. 2022.

observado. Sugerimos, por exemplo, um desenho como o apresentado na Figura 21, o qual você pode utilizar.

Figura 21 Representação esquemática do lançamento vertical.



Fonte: Adaptação pelo autor (2022)[8].

(b) Determine o tempo necessário para a bola atingir o solo (h=0m), após o lançamento. Para isso, aplique a função horária da posição S(t). Qual significado do resultado matemático negativo encontrado?

(c) Explique como seria possível, determinar a altura máxima atingida pela bola, lembrando que a bola não foi lançada da altura zero.

(d) Desenhe o vetor velocidade em dois pontos do lançamento sobre a mesma linha horizontal, sendo um na subida e outro na descida. Indique o sentido pela orientação da seta e os módulos, pelo tamanho do vetor.

[8] Adaptação a partir de imagem disponível em https://proffranca.webnode.com/lancamento-vertical/, acesso em 29 mar. 2022.

(e) Construa o gráfico de s(t) e v(t), partindo da posição 2m e chegando à posição 0m. Indique no gráfico, a classificação do movimento nos intervalos subida e descida, de acordo com o referencial adotado.

3.3.3 Atividade de modelagem computacional

(a) Construa um programa que mostre o lançamento vertical de uma bola. Esta poderá partir da posição 0m, ou você poderá colocar a opção de usuário digitar a altura na entrada, desde que limite os valores de entrada. Nesta animação temos dois referentes, a bola e a Terra. Logo o usuário precisará fornecer também a velocidade de lançamento e o valor da aceleração da gravidade. O gráfico s(t) deverá ser construído ao lado da animação. Deverá ser mostrado na tela, a velocidade de lançamento, a aceleração, o tempo de subida, ou do movimento inteiro e a altura máxima atingida. Neste modelo, será desprezada a resistência do ar.

(b) Faça testes e verifique o que acontece com a altura máxima quando se reduz o valor da aceleração da gravidade (g), mantendo a velocidade de lançamento. E o que acontece com a altura máxima, quando se mantém o g constante e se aumenta a velocidade de lançamento.

(c) Verifique o que acontece com o tempo de subida, quando se mantém a velocidade constante e se diminui o valor da aceleração da gravidade. Verifique, ainda, o que acontece com

o tempo de subida, quando mantém a aceleração da gravidade constante, e se aumenta a velocidade de lançamento.

3.4 Estudo do lançamento oblíquo

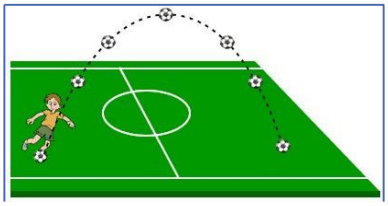
3.4.1 Resumo Teórico

O lançamento oblíquo é um caso de movimento bidimensional. Para o estudo de movimentos bidimensionais, o caráter vetorial da velocidade e da aceleração estão presentes não somente com relação ao sinal (sentido) das grandezas, mas também com relação às suas direções. É importante que antes do estudo deste tipo de movimento, o professor faça o estudo de operações vetoriais com os seus alunos.

Desprezando-se a resistência do ar, o modelo consiste na combinação de dois outros modelos. Na direção vertical atua a aceleração g, sendo assim é o caso de um movimento retilíneo uniformemente variado (MRUV), lançamento vertical. Na direção horizontal, porém, não existe aceleração, resultando assim em um movimento retilíneo e uniforme (MRU). Conforme o Princípio de Galileu (HELOU, GUALTER e NEWTON, 2016), estes dois movimentos ocorrem de forma independente, como se o outro não existisse.

Vamos considerar o exemplo de uma bola sendo chutada por um jogador, em um campo de futebol (Figura 22). Como agora existe uma componente horizontal, além da altura máxima e do tempo em que a bola se projetou no ar, nos interessa ainda saber o alcance máximo que ela atingiria caso tocasse novamente no solo.

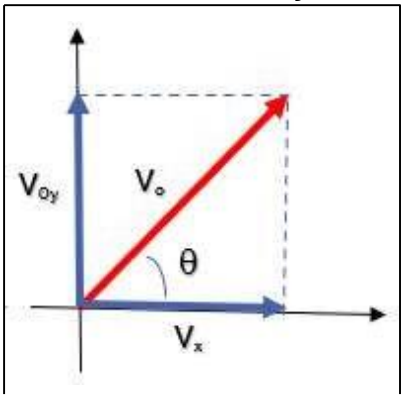
Figura 22 Jogador chutando uma bola em lançamento oblíquo.



Fonte: Ver nota de rodapé[9].

A velocidade de lançamento, que é a velocidade inicial da bola na direção do deslocamento, forma um ângulo com a horizontal. Esta velocidade pode ser decomposta em uma componente horizontal e outra vertical (Figura 23).

Figura 23 Decomposição da velocidade de lançamento no lançamento oblíquo.



Fonte: Elaborado pelo autor (2022).

A posição horizontal do móvel, em qualquer instante, partindo do repouso, é dada por:

$$x = V_x t,$$

onde $V_x = V_0 \cos\theta$.

A posição vertical é dada por:

$$y = y_0 + V_{0y} t - \frac{g t^2}{2}$$

[9] Disponível em https://www.educabras.com/ensino_medio/materia/fisica/mecanica_cinematica/aulas/lancamento_obliquo, acesso em 29 mar. 2022.

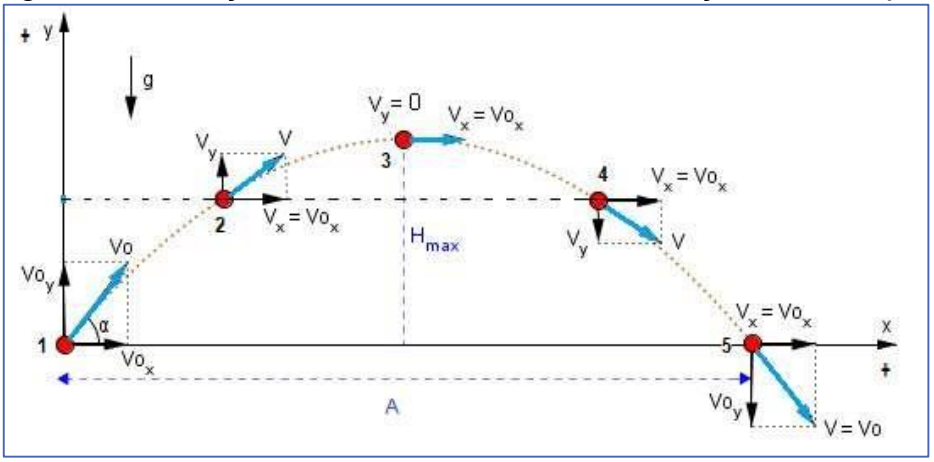
E a velocidade na direção vertical:

$$V_y = V_{oy} - gt$$

onde $V_{oy} = V_0 \operatorname{sen} \theta$.

A Figura 24, dada a seguir, mostra a evolução do vetor velocidade e de duas componentes horizontal e vertical, ao longo de um lançamento oblíquo.

Figura 24 - Variação da velocidade do móvel no lançamento oblíquo.



Fonte: Ver nota de rodapé[10].

Em qualquer ponto da trajetória, a velocidade do móvel é a resultante das duas componentes, V_x e V_y, determinada pela regra do paralelogramo.

$$V_R = \sqrt{V_x^2 + V_y^2}$$

[10] Disponível em https://vamosestudarfisica.com/lancamento-o-que-e-lancamento-obliquo/, acesso em 29 mar. 2022.

A determinação do alcance A, depende do tempo do movimento. No lançamento vertical, verificamos como calcular o tempo para atingir a altura máxima, ou seja, o tempo de subida. O tempo de subida (t_s) é igual ao de descida. Logo, o tempo do movimento (t_m) é duas vezes o tempo de subida.

$$t_s = \frac{V_{0v}}{g} = \frac{V_0 \operatorname{sen}\theta}{g}$$

$$t_m = \frac{2V_0 \operatorname{sen}\theta}{g}$$

Este intervalo de tempo, aplicado à equação da posição em x, nos leva a:

$$x = V_x t$$

$$A = V_0 \cos\theta \frac{(2V_0 \operatorname{sen}\theta)}{g}$$

$$A = \frac{V_0^2 2\cos\theta \operatorname{sen}\theta}{g}$$

Sendo que $2\cos\theta \operatorname{sen}\theta = \operatorname{sen} 2\theta$. De onde chegamos a:

$$A = \frac{V_0^2 \operatorname{sen} 2\theta}{g}$$

A altura máxima, pode ser determinada substituindo-se o tempo de subida t_s na equação da posição em y.

$$y = y_o + V_0 \operatorname{sen}\theta \frac{V_0 \operatorname{sen}}{g} - \frac{g}{2}\left(\frac{V_0 \operatorname{sen}\theta}{g}\right)^2$$

E considerando que a altura é dada por $y - y_o$, temos:

$$H_{máx} = \frac{V_0^2 \operatorname{sen}^2\theta}{2g}$$

3.4.2 Atividade teórica

P.4 Objeto lançado obliquamente

A questão apresenta um modelo de lançamento oblíquo, onde a resistência do ar é desprezada (idealização). Os referentes são o objeto, a Terra. É importante o professor ratificar o fato de que, assim como no lançamento vertical, o referencial estabelecido não muda quando o sentido do objeto passa pelo ponto mais alto. Somente a velocidade, em y, muda de sinal quando o objeto passa pelo ponto mais alto, a velocidade em x e a aceleração da gravidade permanecem como o mesmo sinal durante todo o movimento. Chame a atenção para o fato de que o modelo é idealizado, pois despreza a resistência do ar, ou seja, a bola não fará desvios.

Questão 4. Um jogador de basquete, está a uma distância horizontal de 4m da cesta, quando faz um arremesso. Despreze a resistência do ar e considere que a bola abandona a mão do jogador a uma altura de 3,05m. A cesta de basquete, fica exatamente a 3,05m do chão. Considere que o lançamento é feito sob o ângulo de 60°. Considere g=10m/s².

Figura 25 - Jogador de basquete arremessando uma bola à cesta.

Fonte: Ver nota de rodapé[11].

Este é um modelo que se aproxima muito da realidade, dadas as condições de distância entre o arremessador e a cesta, podendo ser aplicado com uma aproximação aceitável no cálculo do intervalo de tempo. Nas questões abaixo, faça o que se pede:

(a) Faça a representação cinemática para esta situação do lançamento através de um desenho, constando de: 1. Defina um eixo de coordenadas para a análise do movimento; 2. Identifique no desenho as grandezas cinemáticas dadas inicialmente, conforme o enunciado.

(b) Determine a velocidade de lançamento necessária para que seja possível atingir a cesta no local determinado. Considere $\sqrt{3} = 1{,}7$.

[11] Disponível em https://www.preparaenem.com/fisica/lancamento-obliquo.htm, acesso em 29 mar. 2022.

(c) Como você compreende a influência da altura do jogador neste tipo de esporte. Um jogador mais baixo lançaria de uma altura menor. O que seria exigido a mais deste jogador? (lembre-se de que os movimentos são independentes).

(d) Verifique a equação do alcance e responda como a aceleração da gravidade influencia, com relação ao alcance máximo no lançamento.

3.4.3 Atividade de modelagem computacional

Nesta atividade, o professor deverá disponibilizar aos alunos, os códigos fontes das outras aplicações (Apêndice B, Apêndice C, Apêndice D e Apêndice E), para que o aluno possa utilizar como base para fazer os comentários no código fonte do lançamento oblíquo no PyCharm.

(a) No código apresentado no Apêndice F, temos um exemplo de lançamento oblíquo que não foi comentado. Analise este código e coloque os comentários conforme os outros exemplos fornecidos pelo professor.

(b) Selecione na internet, imagens para o projétil e um fundo. Execute o código no PyCharm, realize testes e verifique como se comporta o alcance em função dos ângulos, nos intervalos de 0º a 45º e 45º a 90º. Responda também, em que ângulo é possível ter alcance máximo?

(c) Partindo da equação do alcance, você conseguiria justificar matematicamente a conclusão que chegou sobre o ângulo de lançamento na questão anterior, quando executou o código do lançamento oblíquo?

4. Considerações finais

Este material paradidático propõe exercícios de Física, que foram elaborados, considerando que o professor estudará os conceitos de Modelagem Científica e conduzirá os alunos a raciocinarem a partir deste campo conceitual, distinguindo referentes, idealizações dos modelos e suas aproximações.

Ao compreender o processo de planejamento das animações, e os comandos para programá-la, o professor deve produzir modelos computacionais, que podem ser exercícios do livro didático, para que se capacite a orientar os discentes. Caso exista, na escola, um laboratório de informática, deve o professor incentivar os alunos a desenvolverem suas próprias animações, pois estará presente para dar-lhes suporte.

Este trabalho não substitui o plano de aula do professor, é uma proposta do que pode ser feito, de forma complementar, para desenvolver, nos alunos, o pensamento computacional, a capacidade de pensar de forma algorítmica e ainda ajudá-los a priorizarem concepções científicas, frente a situações científicas.

Esperamos que os professores, do Ensino Médio, se sintam motivados a aplicar este conhecimento em sala de aula, e que os alunos possam tê-lo como ferramenta para solidificação de suas concepções científicas. Acreditamos que a prática, neste tipo de atividade de modelagem computacional, priorizando a dominância do pensamento crítico reflexivo sobre o processo mecânico de reprodução de passos, modificará radicalmente a forma como os discentes enxergam a Física.

Referências

BORGES, L. E. Python para desenvolvedores: aborda Python 3.3. 1. ed. Rio de Janeiro: Novatec Editora, 2014.

BRANDÃO, R. V.; ARAÚJO, I. S.; VEIT, E. A. A Modelagem Científica de fenômenos físicos e o Ensino de Física. Física na Escola, v. 9, n. 1, p. 10-14, 2008.

BRANDÃO, R. V.; ARAÚJO, I. S.; VEIT, E. A. A Modelagem Científica vista como um campo conceitual. Caderno Brasileiro de Ensino de Física, Porto Alegre, 28, dezembro 2011. 507-545.

BRANDÃO, R. V.; ARAÚJO, I. S.; VEIT, E. A. A estratégia de modelagem didático-científica para a conceitualização do real Ensino de Física: um estudo de caso com professores do Ensino Médio. Alexandria: Revista de Educação em Ciência e Tecnologia, v. 12, n. 1, p. 85-110, 2019.

COELHO, F. C. Computação Científica com Python. Petrópolis: Edição do Autor, 2007.

FELTRIN, F. B. Python do Zero a programação orientada a objetos. Edição Kindle. Curitiba: Uniorg, 2019.

GASPAR, A. Cinquenta anos de ensino de física: muitos equívocos, alguns acertos e a necessidade de recolocar o professor no centro do processo educacional. Educação: Revista de Estudos da Educação, Maceió, v. 13, n. 21, p. 71-91, 2004.

GASPAR, A. Problemas conceituais de Física para o Ensino Médio. 1. ed. São Paulo: Livraria da Física, 2018.

HALLIDAY, D.; RESNICK, R.; WALKER, J. Fundamentos de Física. Tradução de Ph.D Ronaldo Sérgio de Biassi. 10. ed. Rio de Janeiro: LTC, v. 1, 2016.

HELOU, R. D.; GUALTER, J. B.; NEWTON, V. B. Tópicos de Física 1--Mecânica: Ensino Médio. 3ª. ed. São Paulo: Saraiva, 2016.

MENEZES, N. N. C. Introdução à programação com Python: Algoritmos e lógica de programação para iniciantes. 2ª. ed. São Paulo: Novatec, 2014.

NUSSENZVEIG, H. M. Curso de Física Básica - Volume 1. 4ª. ed. São Paulo: Blucher, v. 1, 2002.

OLIVEIRA, V.; ARAUJO, I. S.; VEIT, and E. A. Resolução de problemas abertos como um processo de modelagem didático-científico no Ensino de Física. Revista Brasileira de Ensino de Física, v. 42, p. n.p., 29 Junho 2020.

PAPERT, S. A máquina das crianças: repensando a escola na era da informática. Tradução de Sandra Costa. Porto Alegre: Artmed, 2008.

ROSSUM, G. V. O tutorial de Python. Python. Disponível em: https://docs.python.org/pt-br/3/. Acesso em: 28 Dezembro 2021.

SILVA, D. M. Python: História e Ascendência. Revista Programar, v. 59, p. 96-99, 2017.

VALENTE, J. A. Integração do pensamento computacional no currículo da Educação Básica: diferentes estratégias usadas e questões de formação de professores e avaliação do aluno. Revista e-Curriculum, São Paulo, v. 14, n. 3, p. 864-897, 2016.

WING, J. Computational Thinking: what and why. COMMUNICATIONS OF THE ACM, v. 49, n. 3, p. 33-35, Março 2006.

Apêndice A - A Linguagem Python e o Pygame

A.1. A Linguagem Python

A.1.1 Origem

O Python foi criado no final da década de 80, por Guido Van Rossum, um matemático formado pela University of Amsterdam e programador holandês. Enquanto trabalhava como implementador para Lambert Meertens, criador da linguagem ABC, tinha o desenvolvimento da linguagem Python como *hobby.* O nome surgiu pelo fato de ser fã da série Monty Python's (SILVA, 2017, p. 97).

Atualmente, esta linguagem é utilizada por grandes empresas de tecnologia, como Yahoo, Microsoft, Nokia e Disney. A versão 3, lançada em 2008 não possui compatibilidade com as versões anteriores, pois teve o objetivo de consertar as falhas (BORGES, 2014, p. 15) .

A.1.2 Vantagens

Python está entre as melhores linguagens para se aprender programação, por ser de fácil aprendizagem e possuir ferramentas que se adaptam a qualquer propósito. Apesar de ser uma linguagem bem concisa, é clara e objetiva (MENEZES, 2014, p. 24).

O segundo ponto, é tratar-se de um software livre, a partir da versão 2.0 (COELHO, 2007, p. xxi), e poder ser utilizado em vários sistemas operacionais (MENEZES, 2014, p. 25), o que pode ser um fator

aliado aos jovens que ainda não possuem renda para ter assinatura de uma linguagem de programação e determinada arquitetura de computador, com sistema operacional específico.

Outra vantagem é dispor de diversas bibliotecas, com diversas finalidades, o que possibilita ao aprendiz obter grandes resultados em pouco tempo, e com menos erros, o que além de aumentar a produtividade, reduz o tempo de desenvolvimento (MENEZES, 2014, p. 25).

A.1.3 Fundamentos

Existem dois tipos de linguagens, as interpretadas como é o caso do Java, C# e Python e as compiladas estaticamente, como C e C++. No primeiro caso, que é o da linguagem Python, cada linha do código é compilada e executada por vez (COELHO, 2007, p. 28). As linguagens compiladas, por sua vez, precisam transformar todo o programa em código de máquina para depois executar. No caso de uma linguagem interpretada, os erros são percebidos no momento da execução, possibilitando "debugar" o código mais facilmente. Se considerarmos o uso por cientistas, o Python permite que o pesquisador mantenha o foco no objeto estudado, ao invés de se preocupar com correção de detalhes da programação. Além disso, não existem trechos do código inúteis ao raciocínio (COELHO, 2007, p. xviii).

A.1.4 Sintaxe

Um arquivo python é constituído por linhas de código que podem continuar em outra linha pelo símbolo “\” (barra invertida) ou, por

parênteses, colchetes ou chaves conforme cada expressão (BORGES, 2014, p. 22).

Durante a escrita do código, faz-se necessário inserir comentários, por meio do símbolo “#”, que não serão lidos na execução do código. Segue um exemplo na Figura A.1, de um programa para calcular a área de um círculo e o perímetro da circunferência, a partir do raio como entrada.

Figura A.1- Exemplo de código em Python.

```
2
3   import geometria
4   # Dado um círculo de raio r, calcule a área e o perímetro
5
6   r = input("Entre com o raio")
7   area = geometria.areac(r)
8   perimetro = geometria.perimetroc(r)
9   print("O circulo possui área de"  + str(area) + " + u² e comprimento de\
10  "+ str(perimetro)) # aqui temos um exemplo de linha quebrada
```

Fonte: Elaborado pelo autor (2022).

Os blocos são delimitados pela endentação (espaço no começo da linha), e são introduzidos pelos comandos abaixo, seguidos de dois pontos (BORGES, 2014, p. 23-24):

- if/elif/else;
- for/else;
- while/else;
- def;
- try/except/finally/else;
- class;
- with;

```
Sintaxe
    comando expressão:
            Bloco
    comando:
            Bloco
```

A.1.5 Interpretador Python

O interpretador Python funciona como uma shell[12] do Unix[13]. Pode ser executado comando a comando, ou pode ser chamado com nome de um arquivo como argumento, o que, no caso, executa o script contido no arquivo (ROSSUM)

A.1.6 Ambientes de Desenvolvimento de Integrado

As IDEs, sigla para ambiente de desenvolvimento integrado, são editores de texto que possuem funcionalidades para facilitar a escrita do código e testar o seu funcionamento. Qualquer editor de texto pode ser utilizado para a programação, até o bloco de notas do Windows, porém, a IDE reúne um conjunto de ferramentas (FELTRIN, 2019, p. 598), para verificar a sintaxe, compilar, verificar as saídas no prompt de comando, etc...

[12] É um programa que permite a interface entre usuário e computador.
[13] É um sistema operacional criado por Kenneth Thompson.

Embora existam diversas IDEs no mercado, indicaremos a descrita abaixo:

Pycharm:

A empresa responsável por esta plataforma é a JetBrains. Ela possui terminal próprio, ambiente com suporte em tempo real e permite gerenciar ambientes virtualizados por projeto. O Pycharm está disponível em htpps://www.jetbrains.com/pycharm/ (FELTRIN, 2019, p. I.616).

Uma tabela de editores pode ser encontrada em https://wiki.python.org/moin/PythonEditors.

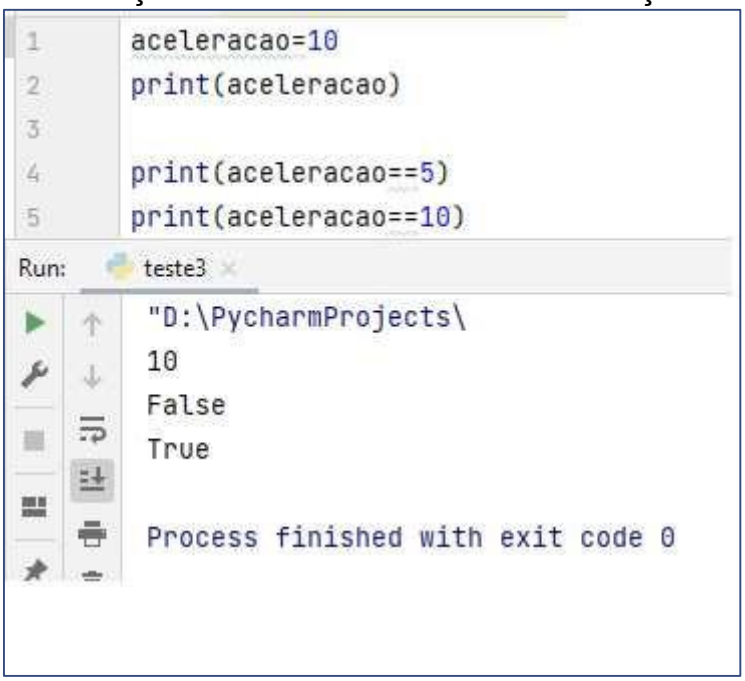
A.1.7 Variáveis

As variáveis são espaços da memória que armazenam valores.

Sintaxe:

```
variável = expressão
```

Figura A.2 - Atribuição de valor a uma variável e verificação do seu valor.



Fonte: Elaborado pelo autor(2022).

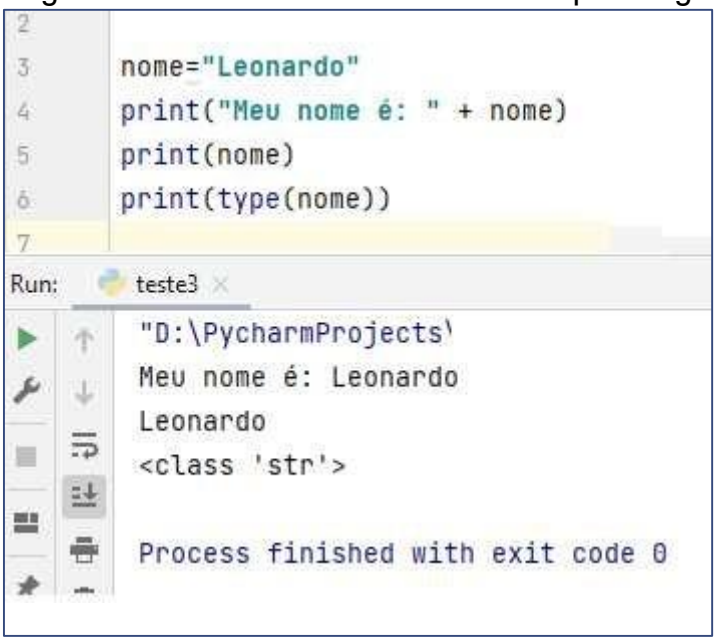
Para verificarmos o valor de uma variável utilizamos o sinal de igual duas vezes.

A.1.8 Tipos de Dados em Python

Os tipos de dados são atribuídos automaticamente pelo interpretador. Eles podem ser do tipo:

- String: São cadeias de caracteres e sua operação de adição consiste em juntar as *strings*.

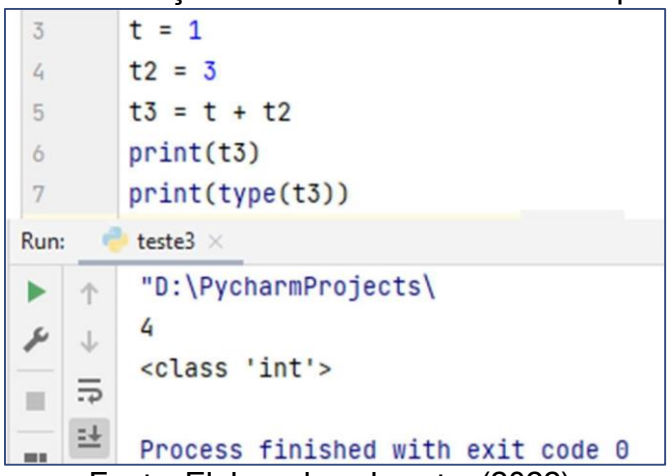
Figura A.3 - Definindo uma variável do tipo string.



Fonte: Elaborado pelo autor (2022).

- Número inteiro: Este tipo de dado é constituído por números inteiros.

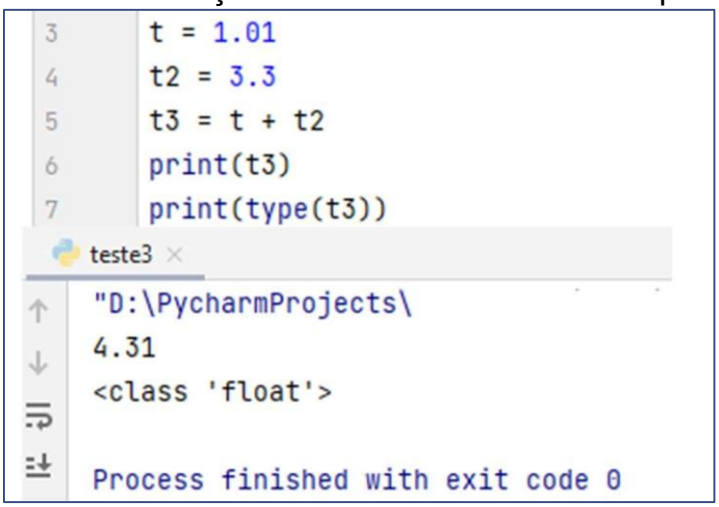
Figura A.4- Atribuição de valor a uma variável do tipo inteiro.



Fonte: Elaborado pelo autor (2022).

- Números de ponto flutuante: São números que contém a parte inteira e a parte decimal. Utiliza-se ponto para a separar as partes.

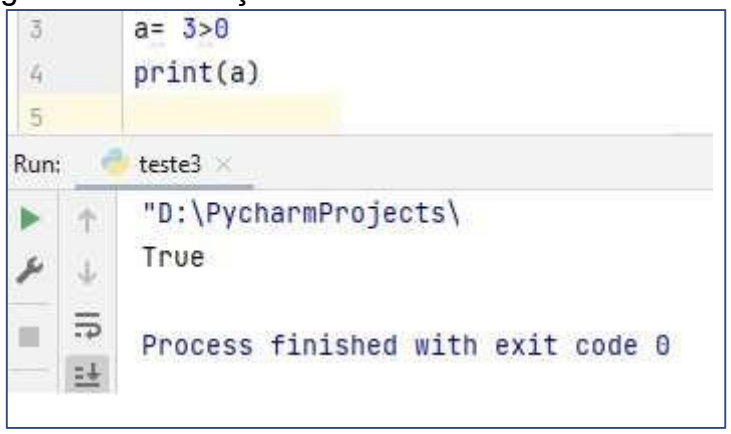
Figura A.5 - Atribuição de valor a uma variável do tipo float.



Fonte: Elaborado pelo autor (2022).

- Boolean: As variáveis do tipo boolean armazenam um valor lógico, de falso ou verdadeiro, "*False*" ou "*True*".

Figura A.6 - Atribuição de valor à uma variável booleana.



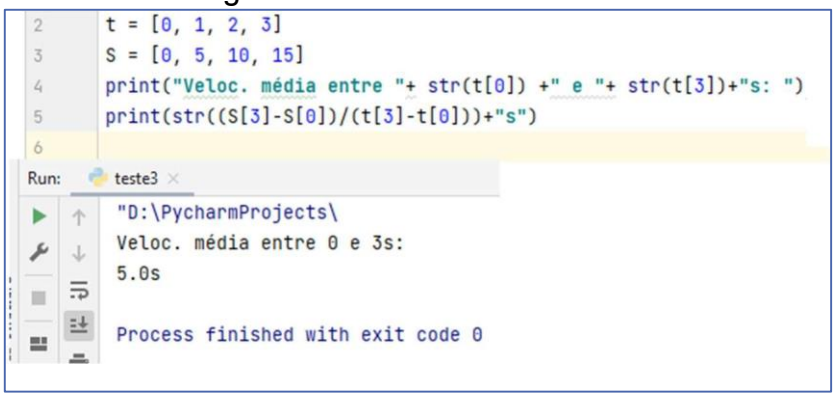
Fonte: Elaborado pelo autor (2022).

A.1.9 Coleções

Em Python, as coleções podem ser organizadas em:

- Listas: É um conjunto de valores organizados por índices, onde o valor entre parênteses é o índice da posição do elemento na lista. Quanto a lista é formada por elementos do mesmo tipo, é chamada de vetor.

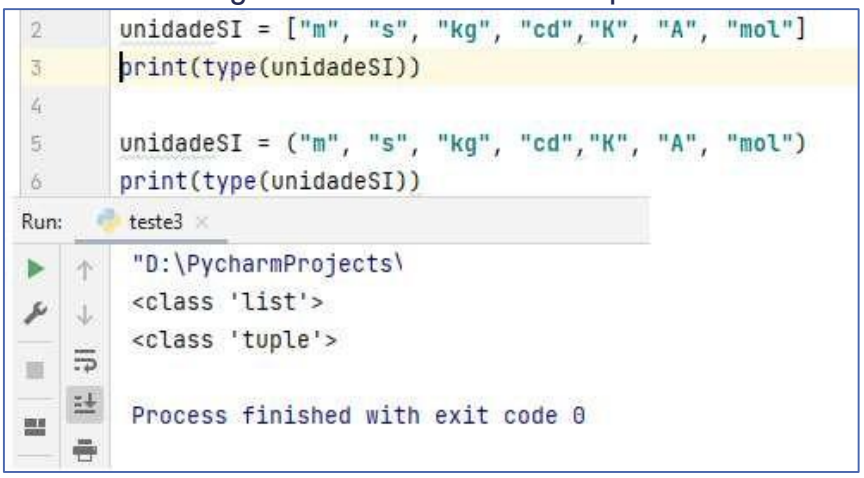
Figura A.7 - Definindo uma lista.



Fonte: Elaborado pelo autor (2022).

- Tuplas: A tupla é uma lista que não pode ser modificada. Ao invés de "[ ]" utiliza-se "( )".

Figura A.8 - Definindo uma tupla.



Fonte: Elaborado pelo autor (2022).

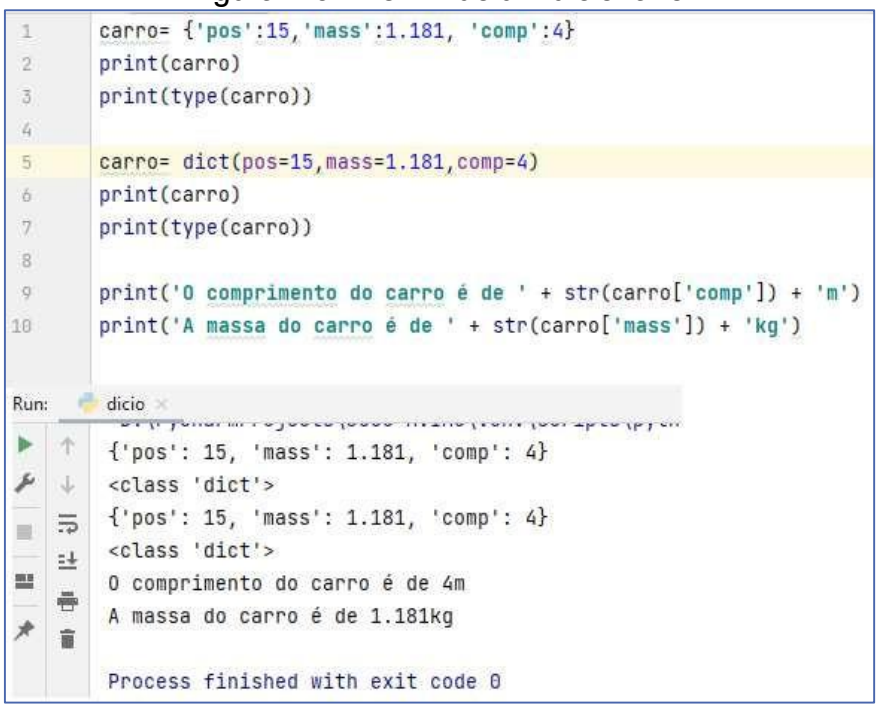
- Dicionários: Os dicionários são coleções que armazenam chave e valor. As chaves são as posições e os valores podem ser: variáveis, listas, tuplas ou dicionários.

Sintaxe:

Por meio da chave, podemos acessar o valor de um dicionário, conforme o exemplo:

- dicio = {'chave':'valor'}

Figura A.9 - Definindo um dicionário.



Fonte: Elaborado pelo autor (2022).

As duas formas como definimos o dicionário "carro" apresentam o mesmo resultado. Mostramos ainda como acessar um valor do dicionário pela chave.

A.1.10 Operadores

Os operadores são utilizados na construção de expressões. Abaixo apresentamos os mais utilizados, que serão utilizados neste trabalho:

- Operadores Aritméticos: São utilizados na execução de operações matemáticas.

Tabela A.1 – Operadores Aritméticos.

Operador	Função	Exemplo
+	Adição	5+2
-	Subtração	5-2
*	Multiplicação	5*2
/	Divisão	4/2
**	Exponenciação	5**2
//	Divisão inteira	5//2
%	Retorna resto da divisão	5%2

Fonte: Elaborado pelo autor (2022).

- Operadores de Atribuição: Atribuem valor a uma variável.

Tabela A 2 - Operadores de Atribuição.

Operador	Função	Exemplo
=	Atribuição de valor à variável	x=2
+=	Incrementa o valor da variável	x +=2 ⇔ x=x+2
-=	Decrementa o valor da variável	x -=2 ⇔ x=x-2
*=	Multiplica o valor de uma variável	x *=2 ⇔ x=x*2
/=	Divide o valor de uma variável	x /=2 ⇔ x=x/2
**=	Eleva o valor de uma variável à uma potência	x **=2 ⇔ x=x**2
%=	Substitui o valor da variável pelo resto da divisão deste valor por um número	x %=2 ⇔ x=x%2

Fonte: Elaborado pelo autor (2022).

- Operadores de Comparação: Comparam valores, tendo um valor booleano como saída.

Tabela A.3 - Operadores de Comparação.

Operador	Função	Exemplo
>	Verifica se o valor da variável é maior do que um número	x>2
<	Verifica se o valor da variável é menor do que um número	x<2
==	Verifica se o valor da variável é igual a um número	x==2
!=	Verifica se o valor da variável difere de um número	x !=2
>=	Verifica se o valor da variável é igual ou maior do que um número	x >=2
<=	Verifica se o valor da variável é igual ou menor do que um número	x <=2

Fonte: Elaborado pelo autor (2022).

- Operadores Lógicos: São conectores que ligam duas expressões condicionais.

Tabela A.4 - Operadores Lógicos.

Operador	Função	Exemplo
and	retorna *True* quando as duas sentenças são verdadeiras	x > 2 and x<4
or	retorna *True* quando, pelo menos, uma das condições é verdadeira	x > 2 or x<4

Fonte: Elaborado pelo autor (2022).

A.1.11 Estruturas Condicionais

As tomadas de decisão do programa ocorrem por meio de estruturas condicionais. Em Python são utilizados "*if*", "*elif*" e "*else*" que significam "*se*", "*mas se*" e "*senão*", respectivamente.

A estrutura condicional está apresentada abaixo.

Sintaxe:

```
if expressão:
        Bloco
elif:
        Bloco
else:
        Bloco
```

A.1.12 Estruturas de Repetição

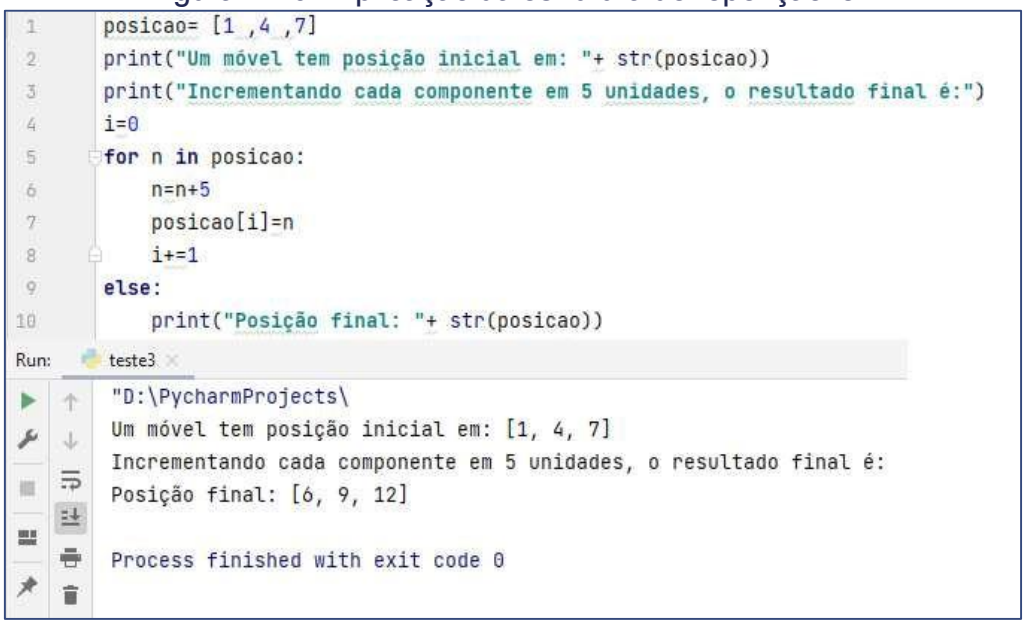
- for: O laço *for* é utilizado quando desejamos percorrer uma lista, gerando iterações para cada item. O comando *else* é opcional, dando a opção de executar um comando quando o valor da variável contido no comando for não se encontra na lista.

Sintaxe:

```
for variável in lista:
        Bloco
else:
        Bloco
```

Apresentamos na Figura A.10, a utilização do laço for para percorrer uma lista que armazena as componentes da posição de um móvel:

Figura A.10 - Aplicação da estrutura de repetição *for*.



Fonte: Adaptação feita pelo autor (2022).

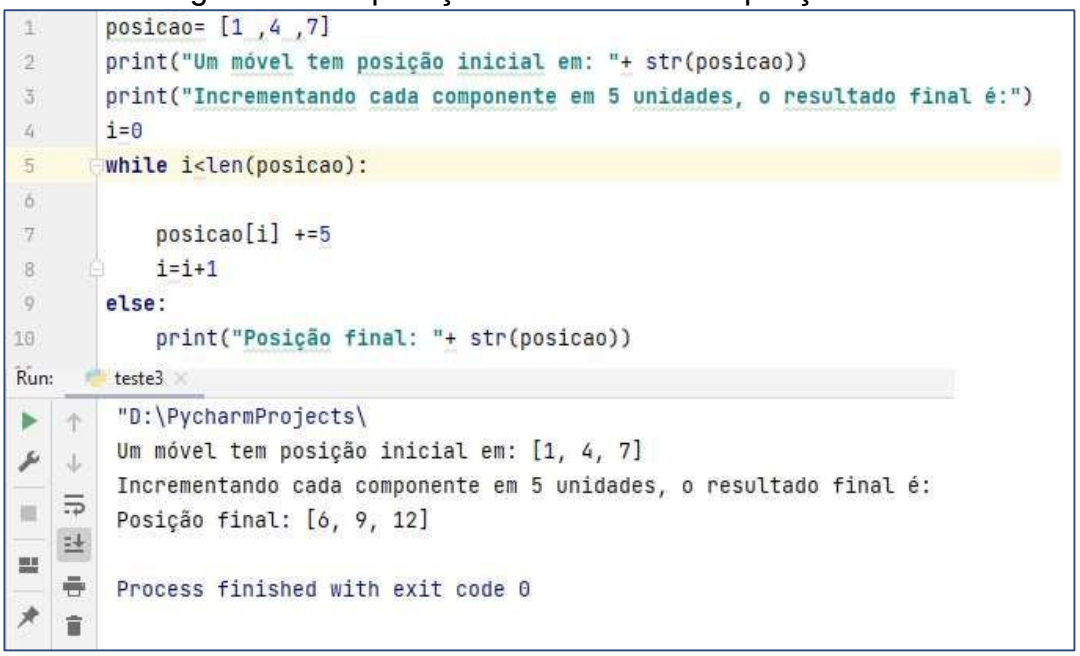
- while: esta estrutura é utilizada para executar comandos enquanto uma condição é satisfeita.

Sintaxe:

```
while condição:
        Bloco
else:
        Bloco
```

O mesmo exemplo do laço *for*, pode ser ajustado para ser executado com *while*, conforme a, produzindo o mesmo resultado:

Figura A 11 – Aplicação da estrutura de repetição while.



Fonte: Adaptação feita pelo autor (2022).

A.1.13 Funções

O objetivo de organizar blocos de comandos em funções, é a possibilidade de chamar esta sequência de comandos várias vezes no código, apenas pelo nome da função que define este bloco de comandos. É uma característica da programação que tem o objetivo evitar repetições.

Sintaxe:

```
def nome da função:
        código da função
```

Na Figura A 12, mostramos um exemplo de função, criada para detectar a colisão de duas imagens no código.

Figura A 12 - Exemplo de função.

```
1  def colisao(lista,valor, intervalo):
2
3      i=0
4      while i < len(lista):
5          z=-intervalo
6          while z<=intervalo:
7              print(-lista[i], abs(valor + z))
8
9              if  lista[i]<abs(valor-z) and lista[i]>abs(valor+z):
10                 print('colidiu')
11
12                 return i
13
14             else:
15                 z=z+1
16             i=i+1
17
18             if i==len(lista):
19                 return None
```

Fonte: Elaborado pelo autor (2022).

Algumas funções são utilizadas de forma rotineira, em qualquer tipo de aplicação, como é caso da função *print()* e da função input(). A função *print(),* mostra o valor do conteúdo entre parênteses no terminal. A função *input()* é utilizada para permitir a entrada de dados pelos usuário, conforme pode ser verificado no exemplo da Figura A.1*Figura* A.*1*- Exemplo de código em Python.

A.2. O Pygame

A.2.1 História

Pete Shinners é um programador que, no ano 2000, estava familiarizado à linguagem C, quando descobriu o Python e a biblioteca SDL (Simple Directmedia Library). A SDL era uma biblioteca escrita em C para controle de recursos multimídia. Com a descontinuidade de desenvolvimento da SDL, Shinners inspirou-se a desenvolver um projeto em Python, conhecido como Pygame (ROSSUM) .

A.2.2 Descrição

O Pygame é uma *game engine* (motor de jogo). Isto significa que esta biblioteca simplifica o desenvolvimento de jogos. Pelo fato de possuir todas as funções visuais e de processamento, o programador se ocupa com outras etapas da construção do jogo, como design e estratégia.

O Pygame pode ser instalado seguindo as instruções disponíveis link https://www.pygame.org/wiki/GettingStarted .

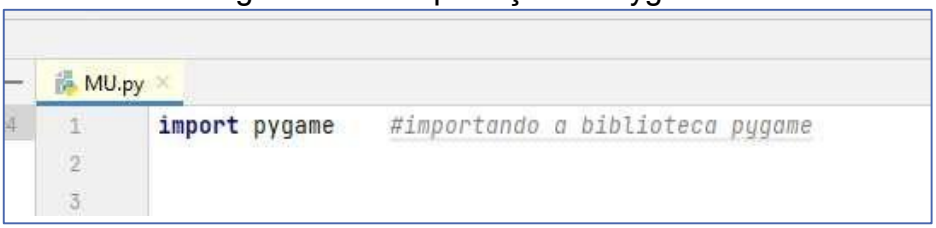
A.2.3 Importação para o código.

Ao criar o arquivo Python, é necessário inicialmente importar o Pygame, que já deverá estar instalado para que a IDE o reconheça.

Para a importação, utiliza-se o comando:

```
import pygame
```

Figura A 13 - Importação do Pygame.



Fonte: Elaborado pelo autor (2022).

A.2.4 Inicialização dos módulos

Além da importação, é necessário inicializar os módulos, o que deve ser feito por meio da função:

Comando:

```
pygame.init()
```

Esta função, que não possui parâmetros, retorna uma tupla (numpass, numfail) que são, respectivamente, o número de módulos inicializados e o números de módulos que falharam.

A.2.5 Módulos

Os módulos em pygame utilizados neste trabalho, estão listados a seguir:

- display: é utilizado para controlar a exibição da janela.

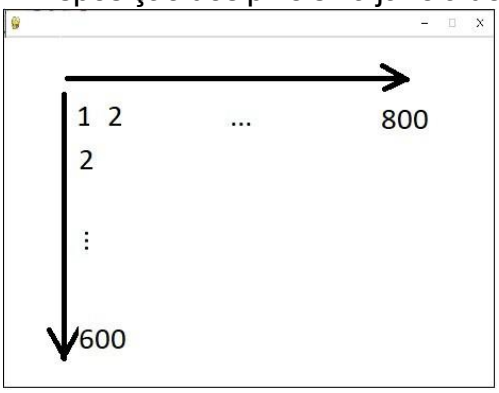
Exemplo:

```
janela = pygame.display.set_mode((800,600))
```

A variável janela armazena uma tela nas dimensões 800 pixels de largura x 600 pixels de altura.

Os pixels na janela de aplicação crescem de cima para baixo e da esquerda para a direita conforme a Figura A.14

Figura A.14- Disposição dos pixels na janela de aplicação.



Fonte: Elaborado pelo autor (2022).

- draw: Possibilita o desenho de formas geométricas em uma superfície.

Exemplo:

```
pygame.draw.line(janela, (0, 0, 0), (310, 51), (310,
214))
```

O módulo *draw* está sendo utilizado para produzir uma linha vertical na superfície janela, na cor preta (0, 0 ,0) na coordenada x= 310, e da coordenada 51 até a 214 em y.

- image: É utilizado para carregar ou salvar imagens.

Exemplo:

```
carro= pygame.image.load('carrinho_direita.png')
```

No exemplo, a imagem do arquivo carrinho_direita.png é armazenado na variável *carro*.
As imagens utilizadas neste caso, estão na mesma pasta da aplicação, caso contrário deverá ser indicado o caminho. É necessário também redimensioná-las, para serem apresentadas na tela em uma proporção coerente. Para redimensionar, basta digitar no *google*: “redimensionar imagem”. Existem várias aplicações online que oferecem este serviço.

- event: É utilizado para detectar e reagir aos eventos realizados pelo usuário, como, por exemplo, pressionar uma tecla ou apertar o "X" da janela de aplicação.

Exemplo:

```
for event in pygame.event.get():
```

```
if event.type==pygame.QUIT:
        janela_aberta= False
```

A condicional `if` verifica se a variável `event` é do tipo `pygame.QUIT` e, se for verdadeira, altera a variável `janela_aberta`, criada anteriormente como TRUE, para o valor FALSE, o que no código, fará com que a janela de aplicação se feche por não atender à condição de repetição do `while`, no trecho de código abaixo:

```
janela_aberta = True
while janela_aberta:
   ## aqui será digitado o código a ser
executado no laço de repetição.
```

- font: É utilizado para o trabalho com fontes de texto. Por exemplo, para a definição do tipo de fonte que será utilizado.

Exemplo:

```
font = pygame.font.Font(None, 30)
```

A variável `font` guarda as informações de uma fonte do tipo None, com tamanho 30.

Mais abaixo, no código, a variável `pos`, recebe através de `font.render()` o texto "Posição: " +str(round((-x),2)) + "m", com *antialias* 1 e cor preta (0,0,0).

Exemplo:

```
   pos = font.render("Posição: " +str(round((-x),2)) +
"m", 1, (0,0,0))
```

- time: Permite o controle do tempo no laço de repetição.

Exemplo:

```
pygame.time.delay(50)
```

No código acima, está sendo estabelecido um intervalo de 50ms entre um quadro e outro, no laço de repetição.

A.2.6 Métodos

Um método é uma função que está associado a uma estrutura de dados, a estrutura de dados que contém o método, é chamada de objeto:

- append: Este método acrescenta elementos a uma lista.

Exemplo:

```
abcissa=[]
abcissa.append(5*tempo)
```

No exemplo, a lista `abcissa` é o objeto.

- fill: a função `set_mode()` foi demonstrada acima no módulo `display`, onde o objeto é do tipo `pygame.Surface`. O método `fill` cobre este objeto com a cor passada como argumento.

Exemplo:

```
janela.fill((255,255,255))
```

- blit: Vimos acima como armazenar um texto em uma variável por meio da função `render()` atrelada a um objeto do tipo `pygame.font.Font`. Para que este texto seja exibido na tela, devemos utilizar o método `blit`:

```
font = pygame.font.Font(None, 30)
pos = font.render("Posição: " +str(round((-x),2)) + "m",
1, (0,0,0))
janela.blit(pos, (20, 60))
```

O texto será exibido nas coordenadas x=20 e y=60.

Tudo o que foi inserido na janela será exibido após a chamada da função `pygame.display.update()`.

Apêndice B - Código Fonte de Simulação Computacional – Algarismo Significativos, Ponto Material e Referencial

```
# Autor: Leonardo Barreto Baptista
# Esta animação trata de um trem atravessando uma ponte
# É necessário adquirir as imagens do trem e do fundo. Podem ser adquiridas no
# site <a href="https://pt.vecteezy.com/vetor-gratis/desenho-animado">Desenho Animado Vetores por Vecteezy</a>.
# as imagens em questão, a imagem do trem foi redimesionada em
# <https://www.resizepixel.com/> para 500x125 pixels e a imagem da ponte 341x95 pixels
# o fundo das imagens foi removido em https://br.depositphotos.com/bgremover.html

import pygame #importando o pygame
pygame.init()    #inicializando o pygame

background=(255,255,255)  #definindo as cores do fundo em RGB  (ver <https://celke.com.br/artigo/tabela-de-cores-html-nome-hexadecimal-rgb>
entretamanhoponte = input("Digite o tamanho da ponte (m):")  #entrada para o usuário digitar o tamanho da ponte.
tamanho_da_ponte=int(entretamanhoponte) #Transforma a entrada em inteiro
fundo = pygame.image.load('ponte.png')  #leitura da imagem da ponte. Armazena na variável fundo.
fundo2 =pygame.image.load('fundo2.png') #leitura da imagem no fundo. Armazena na variável fundo2.
tamanhodaponte = fundo.get_rect().size #A variável tamanhodaponte recebe as
```

```
dimensões da imagem da ponte, em pixels.
[Cp,H]=tamanhodaponte # separando as dimensões em pixels nas variáveis Cp e
H.
fundo=pygame.transform.scale(fundo,(round(tamanho_da_ponte),round(H)))
#modificando o comprimento da imagem em fundo.
tamanhodaponte = fundo.get_rect().size #guardando as novas dimensões em
tamanhodaponte
[Cp,H]=tamanhodaponte # separando as dimensões em pixels nas variáveis Cp e
H.
entretamanhotrem = input("Digite o tamanho do trem (m): ") #entrada para o
usuário digitar o tamanho do trem
tamanho_do_trem=int(entretamanhotrem)  #Transforma a entrada em inteiro
trem = pygame.image.load('trem.png') #leitura da imagem da trem. Armazena
na variável trem.
tamanhodotrem = trem.get_rect().size  #A variável tamanhodaponte recebe as
dimensões da imagem do trem, em pixels.
[Ct,H]=tamanhodotrem  # separando as dimensões em pixels nas variáveis Ct e
H.
trem=pygame.transform.scale(trem,(round(tamanho_do_trem),round(H)))
#modificando o comprimento da imagem trem.
tamanhodotrem = trem.get_rect().size  #guardando as novas dimensões em
tamanhodotrem
[Ct,H]=tamanhodotrem     # separando as dimensões em pixels nas variáveis Ct
e H.

janela= pygame.display.set_mode((1200, 500))  #criando uma variável janela
e dando as dimensões 1200X500 pixels
pygame.display.set_caption("Estudo do movimento de corpos extensos")
#Atribuindo um título à janela

Strem = 0    #definindo como 0, a posição do trem
entrevtrem = input("Digite a velocidade trem (m/s): ") #velocidade do trem
Vtrem=int(entrevtrem)  #transformando a entrada em inteiro
tempo=0 #definindo o instante como zero
font = pygame.font.Font(None, 30)  #A função pygame.font.Font recebe o tipo
e o tamanho da fonte como atributos
# e armazena na variável font
janela_aberta = True     #Criando uma variável booleana janela_aberta como
verdadeira
flag=0      # Esta flag será criada como 0 e mudará para 1 quando o usuário
```

```
quiser iniciar a animação

while janela_aberta:      #A condição de repetição, é janela_aberta ter
valor True

    pygame.time.delay(50)     #Atribuindo um delay de 50ms, entre um loop e
outro

    for event in pygame.event.get():    #Laço for para identificar evento
        if event.type == pygame.QUIT:    #Comparação do tipo de valor em
event com o evento do tipo QUIT
            janela_aberta = False       # Se for igual, mudar a variável
janela_aberta para False

    comandos = pygame.key.get_pressed()     #armazenar a tecla pressionada
na variável comandos


    #Definindo os textos que serão mostrados na Tela de aplicação
    text_Cp = font.render("Tamanho da Ponte: " + str(Cp) + " m", 1, (0, 0,
0))
    text_Ct = font.render("Tamanho da Trem: " + str(Ct) + " m", 1, (0, 0,
0))
    text_st = font.render("Posição do trem (m): " + str(Strem) + " m", 1,
(0, 0, 0))
    text_vt = font.render("Velocidade do trem (m/s): " +
str(round(Vtrem,2)) + " m/s", 1, (0, 0, 0))
    text_t = font.render("Tempo:" + str(round(tempo,2)) + "  s", 1, (0, 0,
0))
    text_msg = font.render("Pressione ESPAÇO para iniciar", 1, (0, 0, 255)
)

    #imprimindo as imagens na superfície janela
    janela.fill(background)
    janela.blit(fundo2, (Ct + Cp, 340))
    janela.blit(fundo2, (0, 340))
    janela.blit(fundo, (Ct, 340))
    janela.blit(trem, (int(Strem), 220))

    # imprimindo os textos na superfície janela
```

```
    janela.blit(text_Cp, (20, 10))
    janela.blit(text_Ct, (20, 30))
    janela.blit(text_st, (20, 50))
    janela.blit(text_vt, (20, 70))
    janela.blit(text_t, (20, 90))

    if comandos[pygame.K_SPACE]:   #condicional para verificar se a tecla
de espaço foi pressionada
        flag=1      # Em caso positivo, modificar o valor da variável flag
para 1
    if flag==1:    #Condicional para verificar se o valor da variável flag
é igual 1
       #Executa o código abaixo em caso positivo
       if Strem < Cp+Ct:    #condicional que verifica se a posição do trem
é menor
           # que o seu comprimento mais o comprimento da ponte
            Strem=Strem+Vtrem/20 #incrementa a posição do trem de 1/20 de
sua velocidade, que
           # é fração de velocidade correspondente à fração de tempo
considerada
            tempo=tempo+0.05 #incrementa o tempo de 50ms

     else:

            janela.blit(text_msg, (20, 120)) #informação para o usuário
iniciar a aplicação


    pygame.display.update()  #atualiza a tela para exibição

pygame.quit()  #saindo do pygame
```

Apêndice C - Código Fonte de Simulação Computacional – Encontro de móveis em movimento retilíneo e uniforme

```
#autor: Leonardo Barreto Baptista
# Esta animação é sobre o encontro de dois móveis realizando MRU
# O gráfico ao lado indica a posição de cada móvel em função do tempo
#após inserir os valores, a animação estará atrás da tela do pycharm
# é necessário providenciar as imagens dos carrinhos e da pista
# os carros têm dimensões 68x36 pixels e a pista, 1295x304 pixels
# O carro A estará à esquerda e o carro B, à direita
# O carro vermelho, deve ficar à esquerda e o azul à direita, para ser
coerente com o gráfico.

import pygame    #importando o pygame

pygame.init()     #inicializando o pygame

entreSB = input("Digite a distância:")  #entrada para o usuário digitar a
posição do carro B
SB0=int(entreSB)  #  A orientação da trajetória é da esquerda para a
direita
#logo, a posição do carrinho B (azul), é a distância entre os móveis

SA = 0 # Definindo que o carrinho vermelho estará na posição zero

entreVA = input("Digite a velocidade do móvel A:")  #entrada para o usuário
digitar a velocidade do carro A
entreVB = input("Digite a velocidade do móvel B:")  #entrada para o usuário
digitar a velocidade do carro B


fundo = pygame.image.load('pista.png')
```

```
janela: None = pygame.display.set_mode((1200, 500))
background=(255,255,255)

# As variáveis abaixo, recebem os valores de entrada, fazendo a conversão
para inteiro
SB=int(entreSB)
VA=int(entreVA)
VB=int(entreVB)

#os vetores abaixo são criados para armazenar as posições e os instantes
ordenadaA = []
ordenadaB = []
abcissa = []


carrinhoA = pygame.image.load('carrinho_verm_sup.png')  #leitura da imagem
do carro A. Armazena na variável carrinhoA.
carrinhoB = pygame.image.load('carrinho_azul_sup.png')  #leitura da imagem
do carro B. Armazena na variável carrinhoB.


font = pygame.font.Font(None, 30)


#Definindo o instante inicial
tempo = 0

pygame.display.set_caption("Encontro de móveis")  #Atribuindo um nome à
janela da aplicação

janela_aberta = True  # definindo uma variável booleana janela como True
flag=0    #flag criada para identificar o encontro
flag2=0  #flag para verificar quando o usuário apertou a tecla espaço,
permitindo iniciar a aplicação

while janela_aberta:      #A condição de repetição, é janela_aberta ter
valor True

        pygame.time.delay(20)     #Atribuindo um delay de 50ms, entre um
loop e outro
```

```
        for event in pygame.event.get():    #Laço for para identificar
evento
            if event.type == pygame.QUIT:     #Comparação do tipo de valor
em event com o evento do tipo QUIT
                janela_aberta = False       # Se for igual, mudar a variável
janela_aberta para False

        comandos = pygame.key.get_pressed()     #armazenar a tecla
pressionada na variável comandos

        # Definindo os textos que serão mostrados na Tela de aplicação
        text_d = font.render("Distância inicial entre os móveis: " +
str(SB0) + " m", 1, (0, 0, 0))
        text_sa = font.render("Posição do móvel vermelho (m): " + str(SA)
+ " m", 1, (0, 0, 0))
        text_sb = font.render("Posição do móvel azul (m): " + str(SB) + "
m", 1, (0, 0, 0))
        text_va = font.render("Velocidade do móvel vermelho (m/s): " +
str(VA) + " m/s", 1, (0, 0, 0))
        text_vb = font.render("Velocidade do móvel azul (m/s): " + str(VB)
+ " m/s", 1, (0, 0, 0))
        text_t = font.render("Tempo:" + str(round(tempo, 2)) + "  s", 1,
(0, 0, 0))
        text_msg = font.render("Pressione ESPAÇO para iniciar", 1, (0, 0,
255))
        ponto_azul = font.render(".", 10, (0, 0, 255))
        ponto_verm = font.render(".", 10, (255, 105, 97))
        text_sm = font.render("x(m)", 5, (255, 105, 97))
        text_ts = font.render("t(s)", 5, (255, 105, 97))


        janela.fill(background)    #Definindo a cor do fundo da Tela de
aplicação

        janela.blit(fundo, (0, 200)) #Imprindo a pista na tela

        # Desenhando as linhas do gráfico
        pygame.draw.line(janela, (0, 0, 0), (800, 51), (800, 110+SB0/4))
        pygame.draw.line(janela, (0, 0, 0), (800, 110+SB0/4), (1200,
```

```
110+SB0/4))

        # Colocando os textos nos eixos do gráfico
        janela.blit(text_sm, (800, 51))
        janela.blit(text_ts, (1100, 120+SB0/4))

        # imprimindo os textos na superfície janela
        janela.blit(text_d, (20, 10))
        janela.blit(text_sa, (20, 30))
        janela.blit(text_sb, (20, 50))
        janela.blit(text_va, (20, 90))
        janela.blit(text_vb, (20, 110))
        janela.blit(text_t, (20, 70))

        # imprimindo os carrinhos na superfície janela
        janela.blit(carrinhoA, (int(SA), 320))
        janela.blit(carrinhoB, (int(SB), 400))

        # adicionando as posições aos carrinhos e o instante
        ordenadaA.append(int(SA))
        ordenadaB.append(int(SB))
        abcissa.append(20*tempo) # a multiplicação por 20, é por causa da
escala do gráfico



        i = 0  # atribuindo o valor zero ao contador da lista.
        while i < len(ordenadaA): # a cada loop, executar os pontos até o
tamanho da lista
            # imprimindo os pontos do gráfico
            janela.blit(ponto_verm, (800 + abcissa[i]/5, 95+SB0/4-
ordenadaA[i]/4))
            janela.blit(ponto_azul, (800 + abcissa[i]/5, 95+SB0/4-
ordenadaB[i]/4))
            i += 1 #incrementando o contador da lista

        if comandos[pygame.K_SPACE]:  # condicional para verificar se a
tecla de espaço foi pressionada
            flag2 = 1  # Em caso positivo, modificar o valor da variável
flag para 1
```

```
        if flag2==1:    #verifica se a flag2 é igual a 1 para iniciar a o
incremento da posição
            if SB > 0: # verifica se o carrinho B ainda não chegou no fim
da pista
                #a fração da velocidade, corresponde à fração de tempo do
loop
                SA=SA+VA/20   #o móvel A tem incremento de sua posição de
VA/20
                SB=SB-VB/20    #o móvel B tem decremento de sua posição de
VB/20
                tempo=tempo+0.05
        else:
             janela.blit(text_msg, (20, 130))  #informação para o usuário
iniciar a aplicação

        if flag==1: #esta flag se modifica para 1 quando a condicional if
abaixo, é satisfeita
            janela.blit(tempo_enc, (posicaox, 120 + SB0/4))#imprime a
posição de encontro no gráfico

        if SB+VB/20>SA>SB-VB/20:  #condição para o encontro dos móveis
             tempo_enc = font.render(str(round(tempo, 2)), 1, (0, 0, 0))
#texto para o tempo encontro
             posicaox=800+2*round(tempo, 2)#definindo a posição do tempo
de encontro no gráfico
             janela.blit(tempo_enc, (posicaox, 120 + SB0/4)) #imprimindo o
tempo de encontro no gráfico
             flag=1  # modificando a variável flag para o valor 1


        pygame.display.update() #atualizando a tela para exibição


pygame.quit()  #saindo do pygame
```

Apêndice D - Código Fonte de Simulação Computacional – Movimento retilíneo uniformemente variado

```
# Nesta animação, um móvel executa MRUV.
# Esta é uma aplicação para o aluno entender o gráfico do moledo teórico, não representa.
# o movimento de uma carro no mundo real, visto que a manobra de mudança de sinal é impossível.
# É necessário adquirir as imagens do carrinho e do fundo. Podem ser adquiridas no
# site https://craftpix.net/freebies/free-fairy-tale-game-backgrounds/, a imagem do carrinho
# foi redimesionada em <https://www.resizepixel.com/> para 90x51 pixels.
# Devem ser geradas duas imagens do carrinho, uma para direita e outra para a esquerda
# o fundo da imagem do carro foi removido em https://br.depositphotos.com/bgremover.html
# O posição é incrementada de Vxdeltat *velocidade vezes tempo) onde deltat tende a zero

import pygame  #importando o pygame

pygame.init()     #inicializando o pygame

janela = pygame.display.set_mode((800,600))    #criando uma variável janela e dando as dimensões 800X600 pixels

pygame.display.set_caption("Movimento Uniformemente Variado")  #Atribuindo um título à janela
```

```
janela_aberta = True  #Criando uma variável booleana janela_aberta como
verdadeira


x=0     # variável x armazena a posição
velocidade=0    # variável velocidade armazena a velocidade instantânea

carro= pygame.image.load('carrinho_direita.png')  #leitura da imagem da
carrinho virado para a direita. Armazena na variável carro.
fundo= pygame.image.load('fundo.png')     #leitura da imagem do fundo.
Armazena na variável fundo.

font = pygame.font.Font(None, 30)    #A função pygame.font.Font recebe o
tipo e o tamanho da fonte como atributos
# e armazena na variável font

tempo=0    #definindo o instante inicial igual a zero

#Criando listas para armazenar as posição do móvel e instante
ordenada=[]
abcissa=[]
flag=0  #flag criada para iniciar a contagem do tempo quando tiver valor
igual a 1

while janela_aberta:    #A condição de repetição, é janela_aberta ter valor
True


        pygame.time.delay(50)  #Atribuindo um delay de 50ms, entre um loop e
outro

        for event in pygame.event.get():  # Laço for para identificar evento
            if event.type == pygame.QUIT:  # Comparação do tipo de valor em
event com o evento do tipo QUIT
                janela_aberta = False  # Se for igual, mudar a variável
janela_aberta para False

        # Definindo os textos que serão mostrados na Tela de aplicação
        pos = font.render("Posição: " +str(round((-x),2)) + "m", 1, (0,0,0))
        text_v = font.render("Velocidade:"+str(round((velocidade),2)) +
```

```
"m/s", 1, (0,0,0))
        text_a = font.render("Aceleração: 0 m/s²", 1, (0,0,0))
        text_t = font.render("Tempo:" + str(round((tempo), 2)) + "s", 1, (0,
0, 0))
        text_sm = font.render("X(m)", 1, (0, 0, 0))
        text_ts = font.render("t(s)", 1, (0, 0, 0))
        ponto = font.render(".",5,(0, 0, 0))

        comandos = pygame.key.get_pressed()     #armazenar a tecla
pressionada na variável comandos

        if x>-190 and x<80:    #condicional if para verificar se a posição do
fundo está no intervalo -190 a 80
            x-=velocidade*0.05     #se a tecla direcional não for
pressionada, a posição do fundo
            # deve ser decrementada de velocidade*delta t. Como o fundo é
que se desloca, não é o carrinho que
            #vai para frente, mas o fundo que vai para trás. O mesmo vale
para o outro sentido do movimento.

        else:

            if x>80:  #verifica se a posição do fundo é maior do que 80 (o
carro está indo para a esquerda).

                velocidade = 0  #Em caso afirmativo, levar a velocidade a zero
                x=78    #e decrementar 2 unidades na posição do fundo (joga o
carrinho para a direita)


            if x<-190:  #verifica se a posição do fundo é menor do que -190
(o carro está indo para a direita).

                velocidade = 0  #Em caso afirmativo, levar a velocidade a zero
                x =-188    #e incrementar 2 unidades  (joga o carrinho para a
esquerda)


        if comandos[pygame.K_RIGHT] and x>=-190:  #condicional verifica se a
tecla direita foi pressionada no intervalo
```

```
            flag=1     # modifica a variável flag para 1
            velocidade = velocidade + 0.05    #Em caso afirmativo, incrementa
a velocidade de aceleração*delta t  (a=1)
            # Se v>0 - progressivo acelerado,    se v<0 - retrógrado retardado
            text_a = font.render("Aceleração: 1 m/s²", 1, (0,0,0))    #Texto
para mostrar aceleração positiva

            if velocidade >=0:  #condicional para verificar se velocidade não
é negativa
                carro= pygame.image.load('carrinho_direita.png')  #Se sim, a
variável carro recebe a imagem carrinho_direta

        if comandos[pygame.K_LEFT] and x <=80:  #condicional verifica se a
tecla esquerda foi pressionada no intervalo
            flag=1  # modifica a variável flag para 1
            velocidade = velocidade - 0.05  #Em caso afirmativo, decrementa
a velocidade de aceleração*delta t  (a=1)
            # Se v>0 - progressivo retardado,    se v<0 - retrógrado acelerado
            text_a = font.render("Aceleração: -1 m/s²", 1, (0, 0, 0))
#Texto para mostrar aceleração negativa

            if velocidade <= 0:    #condicional para verificar se velocidade
não é positiva
                carro = pygame.image.load('carrinho_esquerda.png') #Se sim,
a variável carro recebe a imagem carrinho_esquerda


        ordenada.append(int(x)/2) #armazenando a posição do fundo na lista
ordenada
        abcissa.append(5*tempo)  #armazenando tempo na lista abcissa
        #os fatores de multiplicação são para ajustar a escala do gráfico

        #imprimindo na tela a imagem do fundo
        janela.blit(fundo,(10*x,0))
        janela.blit(fundo, (10*x+990, 0))
        janela.blit(fundo, (10*x-990, 0))
        janela.blit(fundo, (10*x+2*990, 0))

        # imprimindo na tela a imagem do carro e os textos
        janela.blit(carro, (50, 500))
```

```
        janela.blit(pos, (20, 60))
        janela.blit(text_v, (20, 80))
        janela.blit(text_a, (20, 100))
        janela.blit(text_t, (20, 120))
        janela.blit(text_sm, (258, 54))
        janela.blit(text_ts, (608, 217))

        # desenhando na superfície janela, as linhas do gráfico
        pygame.draw.line(janela, (0, 0, 0), (310, 51), (310, 214))
        pygame.draw.line(janela, (0, 0, 0), (310, 214), (610, 214))

        if flag==1:  #verifica se a flag é igual a 1
          tempo = tempo+0.05    # Se positivo, incrementar o tempo

        i=0 # atribuindo o valor 1 ao contador da lista.
        while i<len(ordenada):
            janela.blit(ponto,(310+abcissa[i], 200 +ordenada[i]))  #
imprimindo pontos no gráfico
            i += 1

            if len(ordenada)==1200:    # deleta o primeiro valor da lista
quando o tamanho
                 #da lista é 1200
                del ordenada[1]  # Se sim, deleta o primeito elemento da
lista
                #isso provoca o efeito do gráfico se deslocar para a
esquerda

        pygame.display.update()    #atualizando a tela para exibição

pygame.quit()  #saindo do pygame
```

Apêndice E - Código Fonte de Simulação Computacional – Lançamento Vertical

```
# Nesta animação, um móvel executa MRUV na direção vertical.
# Esta é uma aplicação para o aluno entender o gráfico Sxt no lançamento vertical
# Este movimento é o mais próximo do modelo MRUV.
# É necessário adquirir as imagens da bola e do fundo. Podem ser adquiridas no
# site a href='https://www.freepik.com/vectors/sport-cartoon'>Sport cartoon vector created by brgfx - www.freepik.com</a>/


import pygame    #importando o pygame

pygame.init()    #inicializando o pygame

y0 = 0     #definindo a posição inicial de y em 0
y = y0     #definindo a posição de y como a posição inicial

entreVelocidade = input("Digite a velocidade incial de lançamento (m/s) '9.9':") #entrada
# para a velocidade de lançamento

v0=float(entreVelocidade)  #armazenando o valor convertido para float de
# entreVelocidade na variável v0

v = v0  #definindo a velovidade como a velocidade inicial

ordenada = []    #criando lista para armazenar as posições
abcissa = []    #criando lista para armazenar os instantes

entreAceleracao = input("Digite a aceleração da gravidade local (m/s²) '9.9':") #entrada
```

```
# para a aceleração da gravidade
a=-float(entreAceleracao)  # velocidade de lançamento
hmax = 0  # atribuindo um valor para hmax, para ser chamado na função
font.render()
bola = pygame.image.load('bola.png')  #armazenando a imagem da bola na
variável bola
fundo = pygame.image.load('fundo_campo.jpg')  #armazenando a imagem do
campo na variável fundo
janela: None = pygame.display.set_mode((650, 400))    #criando uma variável
janela e dando as dimensões 650X400 pixels

font = pygame.font.Font(None, 30)  #A função pygame.font.Font recebe o tipo
e o tamanho da fonte como atributos
# e armazena na variável font

#Definindo textos que serão mostrados na tela
text = font.render("Posição: ", 1, (0, 0, 0))
text_v = font.render("Aceleração: -10 m/s²", 1, (0, 0, 0))
text_a = font.render("Aceleração: -10 m/s²", 1, (0, 0, 0))

time = 0 #Definindo o tempo como zero
pygame.display.set_caption("Lançamento vertical")  #Definindo o título da
janela
janela_aberta = True  #Criando uma variável booleana janela_aberta como
verdadeira



while janela_aberta:  #A condição de repetição, é janela_aberta ter valor
True

    # Armazenando os textos que serão mostrados na tela, em variáveis
    pos = font.render("Posição: " + str(round(-y,1)) + " m", 1, (0, 0, 0))
    text_v0 = font.render("Velocidade inicial: " + str(round(v0,1)) + "
m/s", 1, (0, 0, 0))
    text_v = font.render("Veloc. no instante: " + str(round(v,1)) + " m/s",
1, (0, 0, 0))
    text_a = font.render("Aceleração:" + str(a) + " m/s²", 1, (0, 0, 0))
    text_t = font.render("Tempo:" + str(round(time, 1)) + "  s", 1, (0, 0,
0))
```

```
    text_gra = font.render(".", 10, (0, 0, 0))
    text_sm = font.render("y(m)", 5, (0, 0, 0))
    text_ts = font.render("t(s)", 5, (0, 0, 0))
    text_60 = font.render(str(round(-y,1)), 5, (0, 0, 0))
    text_hmax = font.render("Altura máxima: " + str(round(hmax,1))+"m", 5,
(0, 0, 0))
    text_espaco1 = font.render("Clique com o mouse na tela e pressione a
tecla ESPAÇO para executar", 5, (241, 13, 13))
    text_espaco2 = font.render("Pressione a tecla ESPAÇO para reexecutar",
5,
                               (241, 13, 13))

    for event in pygame.event.get():  # Laço for para identificar evento
        if event.type == pygame.QUIT:  # Comparação do tipo de valor em
event com o evento do tipo QUIT
            janela_aberta = False  # Se for igual, mudar a variável
janela_aberta para False

    pygame.time.delay(5) #defini um delay de 5ms para cada loop
    comandos = pygame.key.get_pressed()   #armazenar a tecla pressionada na
variável comandos


    if y>=0.0 and time>0:  # condicional para parar a bola depois do
lançamento

        #imprimindo os textos na superfície janela
        janela.blit(text_hmax, (20, 180))
        janela.blit(fundo, (0, 0))
        janela.blit(pos, (5, 100))
        janela.blit(text_v0, (5, 120))
        janela.blit(text_v, (5, 140))
        janela.blit(text_a, (5, 160))
        janela.blit(text_t, (5, 180))
        janela.blit(bola, (250, 10*y + 220))

        #desenhando as linhas do gráfico
        pygame.draw.rect(janela, (255, 255, 255), [300, 50, 450, 200])
        pygame.draw.line(janela, (0, 0, 0), (310, 51), (310, 214))
        pygame.draw.line(janela, (0, 0, 0), (310, 214), (610, 214))
```

```
        #imprimindo os textos do gráfico
        janela.blit(text_sm, (312, 53))
        janela.blit(text_ts, (610, 213))
        janela.blit(text_60, (307, 200 + (3 * (20*y/ 4))))
        #imprimindo o texto da altura máxima, no final do lançamento
        janela.blit(text_hmax,(5,200))

        #imprimindo o texto para pressionar a tecla espaço
        janela.blit(text_espaco2, (60, 300))


        i = 0
        while i < len(ordenada):    #execute enquando a variável i for menor
do que o tamanho da lista
            janela.blit(text_gra, (310 + 10 * abcissa[i], 200 +
ordenada[i]))  #imprime os pontos do gráfico
            i += 1


            if comandos[pygame.K_SPACE]:  #verifica se a tecla ESPAÇO foi
pressionada
             time=0
             v=v0
             abcissa.clear()
             ordenada.clear()


    else:  #para que a bola suba, na referência de pixels o valor de y
precisa ser negativo
        text_t = font.render("Tempo:" + str(round(time, 1)) + "  s", 1, (0,
0, 0))
        janela.blit(fundo, (0, 0))
        janela.blit(pos, (5, 100))
        janela.blit(text_v0, (5, 120))
        janela.blit(text_v, (5, 140))
        janela.blit(text_a, (5, 160))
        janela.blit(text_t, (5, 180))
        janela.blit(bola, (250, 20*y + 220))
        pygame.draw.rect(janela, (255, 255, 255), [300, 50, 450, 200])
        pygame.draw.line(janela, (0, 0, 0), (310, 51), (310, 214))
```

```
        pygame.draw.line(janela, (0, 0, 0), (310, 214), (610, 214))
        janela.blit(text_sm, (312, 53))
        janela.blit(text_ts, (610, 213))
        janela.blit(text_60, (300, 200 + (20*y)))

        if y == 0:  #verifica se y está em zero, no incío da execução da
aplicação
            janela.blit(text_espaco1, (10, 300))  #imprime a mensagem para
o usuário teclar enter

            if comandos[pygame.K_SPACE]:  #verifica se a tecla ESPAÇO foi
pressionada
                ordenada.append(20 * y)
                abcissa.append(5 * time)
                time = time + 0.01

                ## a aceleração e velocidade estão divididas por 100 por
que  o tempo é incrementado de 0.01
                vant = v / 100  ## guarda a fração de 1/20 da atual
velocidade

                v += a / 100
                y -= +vant  ## o sinal negativo é porque o referencial dos
pixels aponta para baixo no pygame.

        else:
                ordenada.append(20*y)  #acresenta a posição na lista
ordenada
                abcissa.append(5*time)    #acresenta o instante na lista
abcissa
                time = time + 0.01  #incrementa o tempo em 10 ms

                ## a aceleração e velocidade estão divididas por 100 por
que  o intervalo é de 10 ms
                vant = v/100 ## guarda a fração de 1/100 da atual
velocidade

                v += a/100
                y -= +vant   ## o sinal negativo é porque o referencial dos
pixels aponta para baixo no pygame.
```

```
                if v<=0.5 and v>0:     #verifica se a bola está na altura
máxima
                    hmax = -y    #atribui o valor y à altura máxima



    i = 0


    while i < len(ordenada):      #verifica se a variável i é menor do que o
tamanho da lista
        janela.blit(text_gra, (310 + 10 * abcissa[i], 200 + ordenada[i]))
#imprimindo os pontos do gráfico
        i += 1

    pygame.display.update()  #atualizando a tela para exibição

pygame.quit()    # saindo do Pygame
```

Apêndice F – Código Fonte de Simulação Computacional – Lançamento Oblíquo

```
import pygame
import vetorcomponentes
import math

pygame.init()
import time

fundo = pygame.image.load('grama.png')

# O CARRINHO VERMELHO É O A
# O CARRINHO AZUL É O B

angx=30
ang = math.radians(angx)  # transformando em inteiro o ângulo de lançamento
velocidadelanc = 10
velocidade=velocidadelanc

g = 10

[X, Y] = [0, 0]  # definindo a posição X,Y


janela: None = pygame.display.set_mode((1200, 600))
background = (255, 255, 255)

projetil = pygame.image.load('projetil.png')

font = pygame.font.Font(None, 30)

# DEFININDO O TEMPO 0 s como o instante inicial
tempo = 0
```

```
pygame.display.set_caption("Lançamento oblíquo")
janela_aberta = True
flag = 0
vx = velocidadelanc * math.cos(ang)
v0y = velocidadelanc * math.sin(ang)
vy = v0y

inicio = 0
while janela_aberta:

    pygame.time.delay(5)

    comandos = pygame.key.get_pressed()

    #if comandos[pygame.K_SPACE]:
     #   inicio = 1

    if comandos[pygame.K_UP]:
        if angx < 90:
            angx = angx + 5
            ang = math.radians(angx)


    if comandos[pygame.K_DOWN]:
        if angx > 0:
            angx = angx - 5
            ang = math.radians(angx)

    if comandos[pygame.K_RIGHT]:
        if velocidadelanc < 20:
            velocidadelanc +=  1

    if comandos[pygame.K_LEFT]:
        if velocidadelanc > 0:
            velocidadelanc -= 5
```

```
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            janela_aberta = False

    # configuração dos textos
    text_d = font.render("Ângulo de lançamento: " + str(angx) + "° -----
Utilize as teclas up e down para alterar" , 1, (240, 13, 13))
    text_v = font.render("Velocidade de lançamento: " +
str(round(velocidadelanc, 2)) + " m/s ---- Utilize as teclas right e left
para alterar", 1, (240, 13, 13))
    text_vi = font.render("Velocidade instantânea: " +
str(round(velocidade, 2))+ "m/s", 1, (0, 0, 0))

    text_posx = font.render("x(m) : " + str(round(X, 2)) + " m", 1, (0, 0,
0))
    text_posy = font.render("y(m) : " + str(round(-Y, 2)) + " m", 1, (0, 0,
0))
    text_velx = font.render("Vx(m/s) : " + str(round(vx, 2)) + " m/s",
1,(0, 0, 0))
    text_vely = font.render("Vy(m/s) : " + str(round(vy, 2)) + " m/s", 1,
(0, 0, 0))
    text_acele = font.render("g(m/s²) : " + str(g) + " m/s²", 1, (0, 0, 0))
    text_t = font.render("t(s):" + str(round(tempo, 2)) + "  s", 1, (0, 0,
0))
    text_seta1 = font.render("Clique com o mouse na tela e pressione a
tecla espaço para executar", 5,
                             (241, 13, 13))
    text_seta2 = font.render("Pressione a tecla espaço para reexecutar", 5,
(241, 13, 13))

    janela.fill(background)
    #janela.blit(fundo, (0, 445))

    if Y >= 0.0 and tempo > 0:
        janela.blit(fundo, (0, 540))
        janela.blit(text_d, (20, 10))
        janela.blit(text_v, (20, 30))
        janela.blit(text_vi, (20, 50))
        janela.blit(text_posx, (20, 70))
        janela.blit(text_posy, (220, 70))
```

```
            janela.blit(text_velx, (420, 70))
            janela.blit(text_vely, (620, 70))
            janela.blit(text_acele, (20, 90))
            janela.blit(text_t, (220, 90))
            janela.blit(projetil, (50 + int(X) * 20, 20 * Y + 540))
            janela.blit(text_seta2, (60, 300))



            if comandos[pygame.K_SPACE]:
                tempo = 0
                vx = velocidadelanc * math.cos(ang)
                vy = velocidadelanc * math.sin(ang)
                [X, Y] = [0, 0]


        else:
            janela.blit(fundo, (0, 540))
            janela.blit(text_d, (20, 10))
            janela.blit(text_v, (20, 30))
            janela.blit(text_vi, (20, 50))
            janela.blit(text_posx, (20, 70))
            janela.blit(text_posy, (220, 70))
            janela.blit(text_velx, (420, 70))
            janela.blit(text_vely, (620, 70))
            janela.blit(text_acele, (20, 90))
            janela.blit(text_t, (220, 90))
            janela.blit(projetil, (50 + int(X) * 20, 20 * Y + 540))



            vetorcomponentes.resultante(janela, 55 + int(X) * 20, 20 * Y + 545,
    2*vx, 2*vy)

            ## final desenhando os vetores

            if Y == 0:
                janela.blit(text_seta1, (10, 300))

                if comandos[pygame.K_SPACE]:
```

```
                vyant = vy / 100
                vy += -g / 100
                X += vx / 100
                Y -= vyant
                tempo = tempo + 0.01
                velocidade = math.sqrt(math.pow(vx, 2) + math.pow(vy, 2))

                if vy < 0.005 and vy > 0:
                    hmax = -Y

        else:
            vyant = vy / 100
            vy += -g / 100
            X += vx / 100
            Y -= vyant
            tempo = tempo + 0.01
            velocidade = math.sqrt(math.pow(vx, 2) + math.pow(vy, 2))

            if vy < 0.005 and vy > 0:
                hmax = -Y

    pygame.display.update()

pygame.quit()
```

Deverá ser criado um arquivo vetorcomponentes.py, com o código abaixo, para gerar o vetor resultante com as suas componentes.

```
#autor: leonardo Barreto Baptista
#Este arquivo gera o vetor resultante e suas componentes
#No código de origem, deverão ser passados como parâmetros, a superfície, a posição x e y
#da origem do eixo de coordenadas dos vetores, os módulos das componentes em x e y
```

```
import pygame
def resultante(surface, xiniciodosvetores, yiniciodosvetores, vx, vy):

        pygame.init()
## desenhando os vetores
        ##vetory
        if vy > 0:
            sinal=1
        if vy < 0:
            sinal=-1

        pygame.draw.line(surface, (0, 0, 0), (xiniciodosvetores,
yiniciodosvetores), (xiniciodosvetores+2*vx, yiniciodosvetores))
        pygame.draw.line(surface, (0, 0, 0), (xiniciodosvetores,
yiniciodosvetores), (xiniciodosvetores, yiniciodosvetores+2*(-vy)))
        pygame.draw.line(surface, (128, 0, 0), (xiniciodosvetores,
yiniciodosvetores), (xiniciodosvetores+2*vx, yiniciodosvetores + 2 * (-
vy)))

        ## desenhando as setas dos vetores
        ##vetorx
        pygame.draw.line(surface, (0, 0, 255), (xiniciodosvetores + 2 * vx-
5, yiniciodosvetores+5), (xiniciodosvetores + 2 * vx, yiniciodosvetores))
        pygame.draw.line(surface, (0, 0, 255), (xiniciodosvetores + 2 * vx-
5, yiniciodosvetores - 5), (xiniciodosvetores + 2 * vx, yiniciodosvetores))



        pygame.draw.line(surface, (0, 0, 255), (xiniciodosvetores-5,
yiniciodosvetores + 2 * (-vy)+5*sinal),
                         (xiniciodosvetores, yiniciodosvetores + 2 * (-
vy)))
        pygame.draw.line(surface, (0, 0, 255), (xiniciodosvetores + 5,
yiniciodosvetores + 2 * (-vy)+5*sinal),
                         (xiniciodosvetores, yiniciodosvetores + 2 * (-
vy)))

        pygame.draw.line(surface, (128, 0, 0), (xiniciodosvetores + 2 * vx-
5, yiniciodosvetores + 2 * (-vy)), (xiniciodosvetores + 2 * vx,
yiniciodosvetores + 2 * (-vy)))
```

```
        pygame.draw.line(surface, (128, 0, 0), (xiniciodosvetores + 2 * vx,
yiniciodosvetores + 2 * (-vy)+5*sinal),(xiniciodosvetores + 2 * vx,
yiniciodosvetores + 2 * (-vy)))
```