11-Construindo Simulador com Polia Fixa

April 20, 2021

1 Informes iniciais

1. A interface do VPython, juntamente com as suas figuras, aparece na mesma célula de importação do vpython pela primeira vez após o kernel ser reiniciado. Então, após executar a célula para se gerar as figuras, **o usuário irá vizualizá-las na célula de importação da biblioteca**;
2. Caso o usuário queira "limpar" as figuras na interface, é preciso reiniciar o kernel e as saídas, indo no menu em `Kernel -> Restart & Clear Output`, se estiver no Jupyter Notebook;
3. É preciso lembrar de **sempre** executar a célula para importar a biblioteca VPython após reiniciar o kernel, uma vez que ela é a responsável por proporcionar a criação das figuras e afins.

2 Construindo Simulador com Polia Fixa

O sistema se trata de dois blocos ligados por um fio inestensível, que passa por uma polia fixa, tendo um bloco suspenso pelo fio e outro apoiado a uma parede. A partir da massa dos blocos e da altura em que o bloco suspenso se encontra em relação ao solo, é possível caracterizar o estado de movimento do sistema com a sua aceleração (eq. 1). Na simulação, é também possível verificar os vetores do peso e da tração nas cordas.

$$a = \frac{m_b}{m_a + m_b} g \quad (1)$$

Sendo a m_b a massa do bloco suspenso e m_a a massa do bloco apoiado a uma parede.

3 Construção da Interface VPython do Simulador

Para criar a simulação, foram utilizados alguns paralelepípedos (`box`) – blocos, solo, paredes -, cilindros (`cilyder`) - polia e cordas -, setas (`arrow`) - vetores da tração e peso - e uma legenda para a altura do bloco suspenso.

Para além disso, como em algumas situações o movimento ocorre de forma muito rápida, para facilitar a vizualização do fenômeno, foi também utilizada uma variável para controle de tempo (`c`) na função `rate`.

[ ]:
```
from vpython import *

Parede = box(pos=vector(-5,0.5,0),size=vector(10,10,3),color=color.orange)
```

```
Solo = box(pos=vector(-2,-4,0),size=vector(16,1,3),color=color.orange)

#Construção da Polia (Conjunto de Cilindros e dois Box)
polia_a = cylinder(pos=vector(1.7,6.2,0),axis=vector(0,0,0.1), radius=0.6,␣
 ↪color = color.yellow)
polia_b = cylinder(pos=vector(1.7,6.2,0.1),axis=vector(0,0,0.2), radius=0.8,␣
 ↪color = color.white)
polia_c = cylinder(pos=vector(1.7,6.2,-0.2),axis=vector(0,0,0.2), radius=0.8,␣
 ↪color = color.white)
barra_polia_b = box(pos=vector(0.8, 5.7, 0.39),size=vector(2.5,0.3,0.1), axis =␣
 ↪vector(0.5, 0.3, 0),color=color.gray(0.5))
barra_polia_c = box(pos=vector(0.8, 5.7, -0.29),size=vector(2.5,0.3,0.1), axis␣
 ↪= vector(0.5, 0.3, 0),color=color.gray(0.5))

#Cordas
corda_a = cylinder(pos=vector(-7,6.75,0),axis=vector(9,0,0), radius=0.05, color␣
 ↪= color.yellow)
corda_b = cylinder(pos=vector(2.3,5,0),axis=vector(0,1,0), radius=0.05, color =␣
 ↪color.yellow)

#Blocos
massa_a = box(pos=vector(-8,6.5,0),size=vector(2,2,2),color=color.red, mass = 0)
massa_b = box(pos=vector(2.3,4,0),size=vector(2,2,2),color=color.blue, mass = 0)

#Vetores
vetor_a = arrow(pos = massa_a.pos + vec(0,0.25,0), axis = vec(2.5, 0, 0),␣
 ↪shaftwidth=0.15)
vetor_b = arrow(pos = massa_b.pos, axis = vec(0, 2.5, 0), shaftwidth=0.15)
vetor_peso = arrow(pos = massa_b.pos, axis = vec(0, -2.5, 0), shaftwidth=0.15)

#Dados
massa_a.mass = float(input('Insira a Massa do Bloco Suspenso: '))
massa_b.mass = float(input('Insira a Massa do Bloco Suspenso: '))
h = float(input('Insira a Altura do Bloco Suspenso: '))

#Legenda com a Altura
h_label = label(pos=vec(3.3,3,0), text='Altura: y = '+ str(h), xoffset=50,␣
 ↪yoffset=0, space=30, height=16, border=4, font='sans')
g = 9.81 #gravidade
a = massa_b.mass*g/(massa_a.mass+massa_b.mass) #aceleração
t_total = (2*h/a)**(1/2) #tempo total

if (t_total <= 0.5): #Variável de controle do tempo para tempos muito curtos
    c = 5
elif (t_total <= 1):
    c = 10
```

```
elif (t_total <= 2):
    c = 20
elif (t_total <= 3):
    c = 30
elif (t_total <= 4):
    c = 40
elif (t_total <= 5):
    c = 50
else:
    c = 100
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

4 O Movimento

Como dito anteriormente, a lógica de criação de simuladores é a mesma. Nesse caso,utilizou-se a mesma varoável para controle de escala (`e`) - ver tutorial 6-Desenvolvimento de Simulações com o VPython.

[ ]:
```
from vpython import *

Parede = box(pos=vector(-5,0.5,0),size=vector(10,10,3),color=color.orange)
Solo = box(pos=vector(-2,-4,0),size=vector(16,1,3),color=color.orange)

#Construção da Polia (Conjunto de Cilindros e dois Box)
polia_a = cylinder(pos=vector(1.7,6.2,0),axis=vector(0,0,0.1), radius=0.6,␣
 ↪color = color.yellow)
polia_b = cylinder(pos=vector(1.7,6.2,0.1),axis=vector(0,0,0.2), radius=0.8,␣
 ↪color = color.white)
polia_c = cylinder(pos=vector(1.7,6.2,-0.2),axis=vector(0,0,0.2), radius=0.8,␣
 ↪color = color.white)
barra_polia_b = box(pos=vector(0.8, 5.7, 0.39),size=vector(2.5,0.3,0.1), axis =␣
 ↪vector(0.5, 0.3, 0),color=color.gray(0.5))
barra_polia_c = box(pos=vector(0.8, 5.7, -0.29),size=vector(2.5,0.3,0.1), axis␣
 ↪= vector(0.5, 0.3, 0),color=color.gray(0.5))

#Cordas
corda_a = cylinder(pos=vector(-7,6.75,0),axis=vector(9,0,0), radius=0.05, color␣
 ↪= color.yellow)
corda_b = cylinder(pos=vector(2.3,5,0),axis=vector(0,1,0), radius=0.05, color =␣
 ↪color.yellow)

#Blocos
massa_a = box(pos=vector(-8,6.5,0),size=vector(2,2,2),color=color.red, mass = 0)
massa_b = box(pos=vector(2.3,4,0),size=vector(2,2,2),color=color.blue, mass = 0)
```

```
#Vetores
vetor_a = arrow(pos = massa_a.pos + vec(0,0.25,0), axis = vec(2.5, 0, 0),␣
 ↪shaftwidth=0.15)
vetor_b = arrow(pos = massa_b.pos, axis = vec(0, 2.5, 0), shaftwidth=0.15)
vetor_peso = arrow(pos = massa_b.pos, axis = vec(0, -2.5, 0), shaftwidth=0.15)

#Dados
massa_a.mass = float(input('Insira a Massa do Bloco Suspenso: '))
massa_b.mass = float(input('Insira a Massa do Bloco Suspenso: '))
h = float(input('Insira a Altura do Bloco Suspenso: '))

#Legenda com a Altura
h_label = label(pos=vec(3.3,3,0), text='Altura: y = '+ str(h), xoffset=50,␣
 ↪yoffset=0, space=30, height=16, border=4, font='sans')
g = 9.81 #gravidade
a = massa_b.mass*g/(massa_a.mass+massa_b.mass) #aceleração
t_total = (2*h/a)**(1/2) #tempo total

if (t_total <= 0.5): #Variável de controle do tempo para tempos muito curtos
    c = 5
elif (t_total <= 1):
    c = 10
elif (t_total <= 2):
    c = 20
elif (t_total <= 3):
    c = 30
elif (t_total <= 4):
    c = 40
elif (t_total <= 5):
    c = 50
else:
    c = 100

e = h/6.5 #controle de escala
dt=0.01
t=0
while(t < t_total):
    rate (c) #Controle do Tempo
    t+=dt
    #Mudança de Posição em A
    massa_a.pos.x = (((a/2)*(t**2))/e) - 8
    corda_a.pos.x = -7 + (((a/2)*(t**2))/e)
    corda_a.axis.x = 9 - (((a/2)*(t**2))/e)
    vetor_a.pos =  massa_a.pos + vec(0,0.25,0)

    #Mudança de Posição em B
    massa_b.pos.y = 4 - (((a/2)*(t**2))/e)
```

```
    corda_b.pos.y = 5 - (((a/2)*(t**2))/e)
    corda_b.axis.y = 1 + (((a/2)*(t**2))/e)
    vetor_b.pos =  massa_b.pos
    vetor_peso.pos = massa_b.pos
    h_label.text = 'h = ' + str(h - round((a/2)*(t**2), 2))
    h_label.pos=massa_b.pos

#Definindo os Valores Finais dos Vetores
vetor_a.shaftwidth =  0.01
vetor_b.shaftwidth = 0.01
vetor_peso.shaftwidth = 0.01
h_label.text = 'h = 0'
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

5 Associando ao Tkinter

Como dito anteriormente, o Tkinter não é necessariamente uma biblioteca fundamental para o desenvolvimento dos simuladores, mas se destaca como um framework capaz de proporcionar o desenvolvimento de interfaces gráficas de usuário (GUI – Graphic User Interface).

5.1 Criando a interface

Nesse caso, utilizamos a mesma lógica de construção apresentada no tutorial 8-Noções Básicas de Tkinter, criando uma janela que solicite os mesmo dados, mas, agora, em uma GUI. Aqui, foram adicionados alguns novos elementos para dar um melhor visual à GUI:

1. `pady`: Espaçamentos da margem em cima e embaixo;
2. `padx`: Espaçamentos da margem nas laterais;
3. `Frame`: Convencionalmente, antes de inserir elementos na janela, aconcelha-se pôr um `Frame` para que seja possível manipular o plano de fundo com maior facilidade, além de ser possível dividir a janela em diferentes `Frame`'s. Por esse motivo, os elementos são dispostos em cima do Frame;
4. `sticky`: Utilizado para orientar o texto dentro da própria célula - à esquerda (oeste - `W`), à direita (leste - `E`), embaixo (sul - `S`) e em cima (norte `N`). É também possível combinar essas direções para formar outra (ex: nordeste - `N+E`).

[ ]:
```
from vpython import *
from tkinter import *

janela = Tk()
janela.geometry('650x430+400+150')
janela.title('SIMULAÇÕES')
janela['pady']=20
janela['padx']=30

container = Frame(janela, highlightbackground='black', highlightthickness=2)
```

```
container['pady']=20
container['padx']=35
container.grid(row=0, column=0)

lab_title = Label(container, text='SISTEMA COM POLIA FIXA', font='times 20␣
 ↪bold', height=3)
lab_mass_a = Label(container, text='Insira a Massa (kg) do Bloco Suspenso: ',␣
 ↪font='times 14 ', height=2)
lab_mass_b = Label(container, text='Insira a Massa (kg) do Bloco Apoiado: ',␣
 ↪font='times 14 ', height=2)
lab_h = Label(container, text='Insira a Altura de Queda (m): ', font='times 14␣
 ↪', height=2)

lab_title.grid(row=0, column=0, columnspan=4)
lab_mass_a.grid(row=1, column=0, columnspan=2, sticky=W)
lab_mass_b.grid(row=2, column=0, columnspan=2, sticky=W)
lab_h.grid(row=3, column=0, columnspan=2, sticky=W)

ed_mass_b = Entry(container, width=26,font='times 12')
ed_mass_a = Entry(container, width=26,font='times 12')
ed_h = Entry(container, width=26,font='times 12')

ed_mass_a.grid(row=1, column=2, columnspan=2, sticky=W)
ed_mass_b.grid(row=2, column=2,  columnspan=2, sticky=W)
ed_h.grid(row=3, column=2,  columnspan=2, sticky=W)

lb1 = Label(container, text='', height=1, font='times 12 italic')
lb1.grid(row=4, column=1, columnspan=2)

bt = Button(container, text = 'Gerar Simulação', font='times 14 bold', pady=12)
bt.grid(row=5, column=1, columnspan=2, sticky=S)

janela.mainloop()
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

5.2 Capturando os Dados

Nesse caso, utilizamos como `camanda` função `bt_clic` e capturamos os dados com a função `get()`. Como dito anteriormente, a função `get()` captura os dados como caractér, sendo necessário transformá-los em números reais (`float()`).No entanto, caso o usuário digite um caractér, ocorrerá erro na execução da simulação.

É interessante que se faça um *tratamento de erros* por meio da estrutura `try` e `except`. De modo geral, enquanto o programa estiver dentro da estrutura `try`, será tentado executá-lo sem erros; caso dê algum erro, o programa se direcionará para a estrutura `except` sem que seja finalizada a sua execução. Nesse caso, o programa tentará converter os valores dentro do `try` para `float` e, caso

ocorra algum erro, irá para a estrutura **except**, onde será exibida a mensagem de dados inválidos, permitindo ao usuário inseri-los novamente.

[ ]:
```
from vpython import *
from tkinter import *

def bt_clic():
        try:
            massa_a = float(ed_mass_a.get())
            massa_b = float(ed_mass_b.get())
            h = float(ed_h.get())
        except:
            lb1['text']='Valores Informados Inválidos!!!'

janela = Tk()
janela.geometry('650x430+400+150')
janela.title('SIMULAÇÕES')
janela['pady']=20
janela['padx']=30

container = Frame(janela, highlightbackground='black', highlightthickness=2)
container['pady']=20
container['padx']=35
container.grid(row=0, column=0)

lab_title = Label(container, text='SISTEMA COM POLIA FIXA', font='times 20␣
 ↪bold', height=3)
lab_mass_a = Label(container, text='Insira a Massa (kg) do Bloco Suspenso: ',␣
 ↪font='times 14 ', height=2)
lab_mass_b = Label(container, text='Insira a Massa (kg) do Bloco Apoiado: ',␣
 ↪font='times 14 ', height=2)
lab_h = Label(container, text='Insira a Altura de Queda (m): ', font='times 14␣
 ↪', height=2)

lab_title.grid(row=0, column=0, columnspan=4)
lab_mass_a.grid(row=1, column=0, columnspan=2, sticky=W)
lab_mass_b.grid(row=2, column=0, columnspan=2, sticky=W)
lab_h.grid(row=3, column=0, columnspan=2, sticky=W)

ed_mass_b = Entry(container, width=26,font='times 12')
ed_mass_a = Entry(container, width=26,font='times 12')
ed_h = Entry(container, width=26,font='times 12')

ed_mass_a.grid(row=1, column=2, columnspan=2, sticky=W)
ed_mass_b.grid(row=2, column=2,  columnspan=2, sticky=W)
ed_h.grid(row=3, column=2,  columnspan=2, sticky=W)
```

```
lb1 = Label(container, text='', height=1, font='times 12 italic')
lb1.grid(row=4, column=1, columnspan=2)

bt = Button(container, text = 'Gerar Simulação', font='times 14 bold', pady=12,␣
 ↪command=bt_clic)
bt.grid(row=5, column=1, columnspan=2, sticky=S)

janela.mainloop()
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

5.3 Criando a Simulação

Nesse caso, basta inserir a simulação dentro da estrutura `try`, e ele será gerada. Caso queira, também é possível "destruir" a janela durante a execução da simulação com o comando `janela.destroy()`.

[ ]:
```
from vpython import *
from tkinter import *

def bt_clic():
        try:
            Parede = box(pos=vector(-5,0.5,0),size=vector(10,10,3),color=color.
 ↪orange)
            Solo = box(pos=vector(-2,-4,0),size=vector(16,1,3),color=color.
 ↪orange)

            #Construção da Polia (Conjunto de Cilindros e dois Box)
            polia_a = cylinder(pos=vector(1.7,6.2,0),axis=vector(0,0,0.1),␣
 ↪radius=0.6, color = color.yellow)
            polia_b = cylinder(pos=vector(1.7,6.2,0.1),axis=vector(0,0,0.2),␣
 ↪radius=0.8, color = color.white)
            polia_c = cylinder(pos=vector(1.7,6.2,-0.2),axis=vector(0,0,0.2),␣
 ↪radius=0.8, color = color.white)
            barra_polia_b = box(pos=vector(0.8, 5.7, 0.39),size=vector(2.5,0.
 ↪3,0.1), axis = vector(0.5, 0.3, 0),color=color.gray(0.5))
            barra_polia_c = box(pos=vector(0.8, 5.7, -0.29),size=vector(2.5,0.
 ↪3,0.1), axis = vector(0.5, 0.3, 0),color=color.gray(0.5))

            #Cordas
            corda_a = cylinder(pos=vector(-7,6.75,0),axis=vector(9,0,0),␣
 ↪radius=0.05, color = color.yellow)
            corda_b = cylinder(pos=vector(2.3,5,0),axis=vector(0,1,0), radius=0.
 ↪05, color = color.yellow)

            #Blocos
```

```
            massa_a = box(pos=vector(-8,6.5,0),size=vector(2,2,2),color=color.
↪red, mass = 0)
            massa_b = box(pos=vector(2.3,4,0),size=vector(2,2,2),color=color.
↪blue, mass = 0)

            #Vetores
            vetor_a = arrow(pos = massa_a.pos + vec(0,0.25,0), axis = vec(2.5,␣
↪0, 0), shaftwidth=0.15)
            vetor_b = arrow(pos = massa_b.pos, axis = vec(0, 2.5, 0),␣
↪shaftwidth=0.15)
            vetor_peso = arrow(pos = massa_b.pos, axis = vec(0, -2.5, 0),␣
↪shaftwidth=0.15)

            #Dados
            massa_a.mass = float(ed_mass_a.get())
            massa_b.mass = float(ed_mass_b.get())
            h = float(ed_h.get())

            #Legenda com a Altura
            h_label = label(pos=vec(3.3,3,0), text='Altura: y = '+ str(h),␣
↪xoffset=50, yoffset=0, space=30, height=16, border=4, font='sans')
            g = 9.81 #gravidade
            a = massa_b.mass*g/(massa_a.mass+massa_b.mass) #aceleração
            t_total = (2*h/a)**(1/2) #tempo total
            janela.destroy()

            if (t_total <= 0.5): #Variável de controle do tempo para tempos␣
↪muito curtos
                c = 5
            elif (t_total <= 1):
                c = 10
            elif (t_total <= 2):
                c = 20
            elif (t_total <= 3):
                c = 30
            elif (t_total <= 4):
                c = 40
            elif (t_total <= 5):
                c = 50
            else:
                c = 100

            e = h/6.5 #controle de escala
            dt=0.01
            t=0
            while(t < t_total):
```

```
                rate (c) #Controle do Tempo
                t+=dt
                #Mudança de Posição em A
                massa_a.pos.x = (((a/2)*(t**2))/e) - 8
                corda_a.pos.x = -7 + (((a/2)*(t**2))/e)
                corda_a.axis.x = 9 - (((a/2)*(t**2))/e)
                vetor_a.pos =  massa_a.pos + vec(0,0.25,0)

                #Mudança de Posição em B
                massa_b.pos.y = 4 - (((a/2)*(t**2))/e)
                corda_b.pos.y = 5 - (((a/2)*(t**2))/e)
                corda_b.axis.y = 1 + (((a/2)*(t**2))/e)
                vetor_b.pos =  massa_b.pos
                vetor_peso.pos = massa_b.pos
                h_label.text = 'h = ' + str(h - round((a/2)*(t**2), 2))
                h_label.pos=massa_b.pos

            #Definindo os Valores Finais dos Vetores
            vetor_a.shaftwidth =  0.01
            vetor_b.shaftwidth = 0.01
            vetor_peso.shaftwidth = 0.01
            h_label.text = 'h = 0'
        except:
            lb1['text']='Valores Informados Inválidos!!!'

janela = Tk()
janela.geometry('650x430+400+150')
janela.title('SIMULAÇÕES')
janela['pady']=20
janela['padx']=30

container = Frame(janela, highlightbackground='black', highlightthickness=2)
container['pady']=20
container['padx']=35
container.grid(row=0, column=0)

lab_title = Label(container, text='SISTEMA COM POLIA FIXA', font='times 20␣
 ↪bold', height=3)
lab_mass_a = Label(container, text='Insira a Massa (kg) do Bloco Suspenso: ',␣
 ↪font='times 14 ', height=2)
lab_mass_b = Label(container, text='Insira a Massa (kg) do Bloco Apoiado: ',␣
 ↪font='times 14 ', height=2)
lab_h = Label(container, text='Insira a Altura de Queda (m): ', font='times 14␣
 ↪', height=2)

lab_title.grid(row=0, column=0, columnspan=4)
lab_mass_a.grid(row=1, column=0, columnspan=2, sticky=W)
```

```
lab_mass_b.grid(row=2, column=0, columnspan=2, sticky=W)
lab_h.grid(row=3, column=0, columnspan=2, sticky=W)

ed_mass_b = Entry(container, width=26,font='times 12')
ed_mass_a = Entry(container, width=26,font='times 12')
ed_h = Entry(container, width=26,font='times 12')

ed_mass_a.grid(row=1, column=2, columnspan=2, sticky=W)
ed_mass_b.grid(row=2, column=2,  columnspan=2, sticky=W)
ed_h.grid(row=3, column=2,  columnspan=2, sticky=W)

lb1 = Label(container, text='', height=1, font='times 12 italic')
lb1.grid(row=4, column=1, columnspan=2)

bt = Button(container, text = 'Gerar Simulação', font='times 14 bold', pady=12,␣
 ↪command=bt_clic)
bt.grid(row=5, column=1, columnspan=2, sticky=S)

janela.mainloop()
```

5.4 Adicionando Outra Janela

Caso haja o interesse em adicionar uma janela para pôr outras informações na simulação, é possível adicionar um botão à GUI que, ao ser clicado, abre uma outra janela construída por meio da função `ajuda()`. Nesse caso, utilizamos a janela de **ajuda**, contendo uma breve explicação do movimento e da simulação.

[2]:
```
from vpython import *
from tkinter import *

def ajuda():
    newWindow = Toplevel(janela)
    newWindow.geometry('520x490+100+150')
    newWindow.title('AJUDA')
    newWindow['pady']=20
    newWindow['padx']=30
    lab_title2 = Label(newWindow, text='SISTEMA COM POLIA FIXA', font='times 20␣
 ↪bold', height=3)
    lab_title2.grid(row=0, column=0)

    a = 'O sistema se trata de dois blocos ligados por um fio inestensível, '
    a = a + 'que passa por uma polia fixa, tendo um bloco suspenso pelo fio e␣
 ↪outro'
    a = a + 'apoiado a uma parede. A partir da massa dos blocos e da altura em␣
 ↪que o bloco '
    a = a + 'suspenso se encontra em relação ao solo, é possível caracterizar o␣
 ↪estado '
```

```
    a = a + 'de movimento do sistema com a sua aceleração. '
    a = a + '\n\nNa simulação, é também possível verificar os vetores do peso e␣
↪da tração nas cordas.'
    lab_desc = Label(newWindow, font='times 12 ',text=a,  wraplength=450)
    lab_desc.grid(row=1, column=0)

def bt_clic():
        try:
            Parede = box(pos=vector(-5,0.5,0),size=vector(10,10,3),color=color.
↪orange)
            Solo = box(pos=vector(-2,-4,0),size=vector(16,1,3),color=color.
↪orange)

            #Construção da Polia (Conjunto de Cilindros e dois Box)
            polia_a = cylinder(pos=vector(1.7,6.2,0),axis=vector(0,0,0.1),␣
↪radius=0.6, color = color.yellow)
            polia_b = cylinder(pos=vector(1.7,6.2,0.1),axis=vector(0,0,0.2),␣
↪radius=0.8, color = color.white)
            polia_c = cylinder(pos=vector(1.7,6.2,-0.2),axis=vector(0,0,0.2),␣
↪radius=0.8, color = color.white)
            barra_polia_b = box(pos=vector(0.8, 5.7, 0.39),size=vector(2.5,0.
↪3,0.1), axis = vector(0.5, 0.3, 0),color=color.gray(0.5))
            barra_polia_c = box(pos=vector(0.8, 5.7, -0.29),size=vector(2.5,0.
↪3,0.1), axis = vector(0.5, 0.3, 0),color=color.gray(0.5))

            #Cordas
            corda_a = cylinder(pos=vector(-7,6.75,0),axis=vector(9,0,0),␣
↪radius=0.05, color = color.yellow)
            corda_b = cylinder(pos=vector(2.3,5,0),axis=vector(0,1,0), radius=0.
↪05, color = color.yellow)

            #Blocos
            massa_a = box(pos=vector(-8,6.5,0),size=vector(2,2,2),color=color.
↪red, mass = 0)
            massa_b = box(pos=vector(2.3,4,0),size=vector(2,2,2),color=color.
↪blue, mass = 0)

            #Vetores
            vetor_a = arrow(pos = massa_a.pos + vec(0,0.25,0), axis = vec(2.5,␣
↪0, 0), shaftwidth=0.15)
            vetor_b = arrow(pos = massa_b.pos, axis = vec(0, 2.5, 0),␣
↪shaftwidth=0.15)
            vetor_peso = arrow(pos = massa_b.pos, axis = vec(0, -2.5, 0),␣
↪shaftwidth=0.15)

            #Dados
```

```
            massa_a.mass = float(ed_mass_a.get())
            massa_b.mass = float(ed_mass_b.get())
            h = float(ed_h.get())

            #Legenda com a Altura
            h_label = label(pos=vec(3.3,3,0), text='Altura: y = '+ str(h),␣
↪xoffset=50, yoffset=0, space=30, height=16, border=4, font='sans')
            g = 9.81 #gravidade
            a = massa_b.mass*g/(massa_a.mass+massa_b.mass) #aceleração
            t_total = (2*h/a)**(1/2) #tempo total
            janela.destroy()

            if (t_total <= 0.5): #Variável de controle do tempo para tempos␣
↪muito curtos
                c = 5
            elif (t_total <= 1):
                c = 10
            elif (t_total <= 2):
                c = 20
            elif (t_total <= 3):
                c = 30
            elif (t_total <= 4):
                c = 40
            elif (t_total <= 5):
                c = 50
            else:
                c = 100

            e = h/6.5 #controle de escala
            dt=0.01
            t=0
            while(t < t_total):
                rate (c) #Controle do Tempo
                t+=dt
                #Mudança de Posição em A
                massa_a.pos.x = (((a/2)*(t**2))/e) - 8
                corda_a.pos.x = -7 + (((a/2)*(t**2))/e)
                corda_a.axis.x = 9 - (((a/2)*(t**2))/e)
                vetor_a.pos =  massa_a.pos + vec(0,0.25,0)

                #Mudança de Posição em B
                massa_b.pos.y = 4 - (((a/2)*(t**2))/e)
                corda_b.pos.y = 5 - (((a/2)*(t**2))/e)
                corda_b.axis.y = 1 + (((a/2)*(t**2))/e)
                vetor_b.pos =  massa_b.pos
                vetor_peso.pos = massa_b.pos
                h_label.text = 'h = ' + str(h - round((a/2)*(t**2), 2))
```

```
                h_label.pos=massa_b.pos

            #Definindo os Valores Finais dos Vetores
            vetor_a.shaftwidth =  0.01
            vetor_b.shaftwidth = 0.01
            vetor_peso.shaftwidth = 0.01
            h_label.text = 'h = 0'
        except:
            lb1['text']='Valores Informados Inválidos!!!'

janela = Tk()
janela.geometry('650x430+400+150')
janela.title('SIMULAÇÕES')
janela['pady']=20
janela['padx']=30

container = Frame(janela, highlightbackground='black', highlightthickness=2)
container['pady']=20
container['padx']=35
container.grid(row=0, column=0)

lab_title = Label(container, text='SISTEMA COM POLIA FIXA', font='times 20␣
 ↪bold', height=3)
lab_mass_a = Label(container, text='Insira a Massa (kg) do Bloco Suspenso: ',␣
 ↪font='times 14 ', height=2)
lab_mass_b = Label(container, text='Insira a Massa (kg) do Bloco Apoiado: ',␣
 ↪font='times 14 ', height=2)
lab_h = Label(container, text='Insira a Altura de Queda (m): ', font='times 14␣
 ↪', height=2)

lab_title.grid(row=0, column=0, columnspan=4)
lab_mass_a.grid(row=1, column=0, columnspan=2, sticky=W)
lab_mass_b.grid(row=2, column=0, columnspan=2, sticky=W)
lab_h.grid(row=3, column=0, columnspan=2, sticky=W)

ed_mass_b = Entry(container, width=26,font='times 12')
ed_mass_a = Entry(container, width=26,font='times 12')
ed_h = Entry(container, width=26,font='times 12')

ed_mass_a.grid(row=1, column=2, columnspan=2, sticky=W)
ed_mass_b.grid(row=2, column=2,  columnspan=2, sticky=W)
ed_h.grid(row=3, column=2,  columnspan=2, sticky=W)

lb1 = Label(container, text='', height=1, font='times 12 italic')
lb1.grid(row=4, column=1, columnspan=2)
```

```
bt = Button(container, text = 'Gerar Simulação', font='times 14 bold', pady=12,␣
 ↪command=bt_clic)
bt.grid(row=5, column=1, columnspan=2, sticky=S)
buttonExample = Button(container, text='Ajuda', font='times 12 bold',␣
 ↪command=ajuda)
buttonExample.grid(row=5, column=0, columnspan=2, sticky=W)

janela.mainloop()
```

Apesar de se diferenciar quanto às equações e formas utilizadas, todos os outros simuladores seguem essa mesma lógica. Ou seja, compreendendo a lógica de construção dos simuladores, juntamente com o conhecimento em Física, é possível se desenvolver uma série de outros simuladores.

[ ]: