Relatório do Trabalho de Física

Igor Patrício Michels
Isaque Vieira Machado Pim

Outubro de 2019

1 Introdução

O trabalho proposto pelo professor constitui na elaboração de um modelo físico com o objetivo de simular um fenômeno físico. Para tanto, nós escolhemos simular um jogo de Minigolfe, no qual o usuário pode interagir com a simulação, como se estivesse jogando uma partida de minigolfe.

2 Objetivo

O objetivo do trabalho é modelar de maneira simples um jogo de minigolfe, utilizando de medidas e dimensões reais, além de apresentar o resultado de maneira interativa, onde o usuário poderá interagir com a física das tacadas.

3 Material

Para o desenvolvimento do trabalho, utilizamos a plataforma Processing com o modo Python para realizar a parte da computação gráfica. Todos os métodos de iteração são baseados no conteúdo apresentado em sala de aula.

4 Metodologia

4.1 Movimentação

A movimentação foi implementada pelo Método de Euler. Com dados da velocidade e da posição calculamos a próxima posição baseada no intervalo de tempo entre os frames do programa.

4.2 Colisões

Para as colisões não foi feito nenhum trabalho sobre conservação de energia. Todas as colisões são perfeitamente elásticas, ou seja, há conservação de energia total. A detecção de colisão é feita iterando-se sobre uma lista de lados onde os resultados de operações com o produto escalar que ligam o centro da bola às extremidades dos lados verificam a colisão.

4.3 Atrito

A modelagem de atrito para objetos rolantes sobre superfícies deformáveis é um assunto muito complexo. Para dar mais realismo à simulação estipulamos o atrito constante baseado em um artigo da Universidade de Berlim sobre futebol de robôs, onde são apresentados resultados empíricos sobre bolas de golfe rolando em carpete. No fim acabamos por adotar um resultado similar, para dar mais conforto ao usuário, baseado em proporções com a velocidade da bola.

5 Desenvolvimento do Código

Aqui será feito apenas o detalhamento do código que gera a física do jogo. As partes de detalhamento visual não serão comentadas.
A bola foi implementada como uma classe. Essa classe contém as características essenciais da bola como a posição, a velocidade e o raio. Além disso, contém os métodos de mover a bola e de acelerá-la. *inserir código da bola e explicar o método de euler ali
Colisão e detecção de colisão foram implementadas como funções que recebem como argumentos objetos da classe bola.

```
#definicoes das classes
class Bola:
    def __init__(self,p,v,r):
        self.v = v.copy()
        self.p = p.copy()
        self.r = r
        self.para = True
    def move(self,dt):
        self.p.add(self.v*dt)

    def xlr8(self,dt):
        v = self.v.copy()
        if self.para:
            if v.mag() < 0.009:
                b.v.sub(b.v)
        if v.mag() < 0.1:
            mi = -0.3*v.mag()
            v.mult(mi)
            self.v.add(v)
        else:
            mi = -0.3*v.mag()**2
            v.mult(mi)
            self.v.add(v)
        self.para = True


def colide(b, s):
    s.normalize()
    vy = s*(b.v.dot(s))
    vx = b.v - vy


    vy = vy* (-1)
    b.p.add(2*vy)
    b.v = vx + vy


def detecta_colisao():
    global lados, b, r2

    for l in lados:
        p1 = l[0].copy()
        p2 = l[1].copy()
        p = p2-p1
        ball = b.p.copy()
        v1 = p.dot(ball-p1)
        v2 = p.dot(ball-p2)

        if v1 > 0 and v2 < 0:
            a_ = (ball-p1) - (v1/(v1-v2))*p
            if a_.x**2 + a_.y**2 <= r2:
                colide(b,a_)


def detecta_buraco():
    global tempo, fase, estado, Menu, t, lados, b, lados_r, ret_r1, ret_r2, buraco,
        Creditos
```

```
    bru = buraco[0]
    if (bru.x-b.p.x)**2 + (bru.y - b.p.y)**2 <= (buraco[1]/2)**2:
        b.p.x = bru.x
        b.p.y = bru.y
        b.v.x = 0
        b.v.y = 0
        tempo+=1
        if tempo >100:
            tempo = 0
            if fase == 1:
                b.p.x = 170
                b.p.y = 300
                b.v.x = 0
                b.v.y = 0
                lados = lados_r
                buraco = buraco_r
            elif fase == 2:
                b.p.x = 170
                b.p.y = 300
                b.v.x = 0
                b.v.y = 0
                lados = lados_m
                buraco = buraco_m
            elif fase == 3:
                b.p.x = 305
                b.p.y = 175
                b.v.x = 0
                b.v.y = 0
                lados = lados_m2
                buraco = buraco_m2
            elif fase == 4:
                estado = Creditos
                fase = 0
                b = Bola(p.copy(),v.copy(),r)
                lados = lados1
                buraco = [PVector(500,300), 7]


            fase += 1


def Minigolf():
    global estado, Menu, t, lados, b, lados_r, ret_r1, ret_r2, buraco
    oldt = t
    t = millis()
    dt = t-oldt

    def fase1():
        rectMode(CORNER)
        stroke(122,166,56,127)
        fill(122,166,56,127)
        rect(250,250,300,100)

        stroke(0)
        for i in lados:
            line(i[0].x,i[0].y,i[1].x,i[1].y)
        fill(0)
        ellipse(buraco[0].x, buraco[0].y, buraco[1], buraco[1])

    def fase2():
        rectMode(CORNER)
        stroke(122,166,56,127)
        fill(122,166,56,127)
        rect(150,255,500,90)

        noStroke()
        fill(149,191,59,100)
        rect(ret_r1[0].x, ret_r1[0].y,ret_r1[1], ret_r1[2])
        fill(149,191,59,255)
```

```
        rect(ret_r2[0].x, ret_r2[0].y,ret_r2[1], ret_r2[2])

        stroke(0)
        for i in lados:
            line(i[0].x,i[0].y,i[1].x,i[1].y)
        fill(0)
        ellipse(buraco[0].x, buraco[0].y, buraco[1], buraco[1])

        if b.p.x > 300 and b.p.x < 350 :
            b.para = False
            b.v.x -= 0.0001*dt
            b.v.y -= 0.0001*dt

        if b.p.x > 450 and b.p.x < 500 :
            b.para = False
            b.v.x += 0.0001*dt
            b.v.y += 0.0001*dt

    def fase3():
        rectMode(CORNER)
        stroke(122,166,56)
        fill(122,166,56)
        rect(100,255,600,90)
        rect(400,155,90,290)
        triangle(590,255,617.5,255-55/2*3**(1/2),700,255)
        triangle(617.5,255-55/2*3**(1/2),672.5,255-55/2*3**(1/2),700,255)
        triangle(700,345,672.5,345+55/2*3**(1/2),617.5,345+55/2*3**(1/2))
        triangle(617.5,345+55/2*3**(1/2),590,345,700,345)

        stroke(0)
        for i in lados:
            line(i[0].x,i[0].y,i[1].x,i[1].y)

        fill(0)
        ellipse(buraco[0].x, buraco[0].y, buraco[1], buraco[1])
        fill(252,237,20)
        triangle(410,270,410,310,480,270)
        triangle(410,330,480,330,480,290)

    def fase4():
        rectMode(CORNER)
        stroke(122,166,56,127)
        fill(122,166,56,127)
        rect(260,155,90,200)
        rect(450,155,90,200)

        stroke(242,188,121)
        fill(242,188,121)
        triangle(261,355,350,445,450,445)
        triangle(450,445,540,355,261,355)

        stroke(0)
        for i in lados:
            line(i[0].x,i[0].y,i[1].x,i[1].y)

        fill(0)
        ellipse(buraco[0].x, buraco[0].y, buraco[1], buraco[1])

        #freiando caso esteja na areia
        if b.p.y > 355:
            b.xlr8(dt)

    background(34,115,55)

    if fase == 1:
        fase1()
    elif fase == 2:
        fase2()
    elif fase == 3:
```

```
195             fase3()
        elif fase == 4:
197             fase4()

199         rectMode(CENTER)
        stroke(0)
201         fill(0)
        text('Exit', 0.11*width, 50)
203         if mouseX <= 0.11*width + 0.1*width and mouseX >= 0.11*width - 0.1*width and mouseY
            <= 50+30 and mouseY >= 50-30:
                noFill()
205             rect(0.11*width, 50, 0.2*width, 60)

207         text('Restart', 0.11*width, 120)
        if mouseX <= 0.11*width + 0.1*width and mouseX >= 0.11*width - 0.1*width and mouseY
            <= 120+30 and mouseY >= 120-30:
209             noFill()
                rect(0.11*width, 120, 0.2*width, 60)
211


213         b.move(dt)
        b.xlr8(dt)
215
        detecta_colisao()
217         detecta_buraco()

219         fill(255)
        stroke(255)
221         ellipse(b.p.x,b.p.y, 2*b.r, 2*b.r)


223
    def setup():
225         size(800, 600)

227 def draw():
        global estado
229         estado()

231 def mouseDragged():
        global inc, ver
233
        if b.v.mag() == 0: #condicao para nao aceitar tacadas se a bola estiver se movendo
235             if sqrt((mouseX - b.p.x)**2 + (mouseY-b.p.y)**2) <= r+10:
                    ver = True
237
                if ver:
239                 inc.x = b.p.x - mouseX
                    inc.y = b.p.y - mouseY
241                 stroke(255,0,0)
                    line(b.p.x,b.p.y,mouseX,mouseY)
243
    def mouseReleased():
245         global inc, ver, estado, Minigolf
        if estado == Minigolf:
247             if ver:
                    b.v.add(inc/300)
249                 inc = PVector(0,0)
                    ver = False
```

No código acima, foram omitidas as definições das variáveis e das características de cada pista: segmentos que delimitam os lados, as posições dos buracos e as posições iniciais das bolas. Além de outras informações cuja omissão não afeta o entendimento do código em si.