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.

5.1 Classe bola

Definição da classe boa, nela define-se que a bola recebe um vetor posição (posição X e posição Y), um vetor velocidade (velocidade em X e velocidade em Y) e o raio, que no trabalho foi definido igual a dois, seguindo o padrão internacional.
Nessa classe, define-se a movimentação da bolinha, adicionando o produto da velocidade com o intervalo de tempo na posição, e a aceleração, a qual sofre uma pequena diminuição em virtude do atrito dado.

```
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
```

5.2 Funções para colisão

Nessa parte, estamos definindo as funções que calculam as colisões da bola junto aos lados da pista e/ou obstáculos. Lembramos que a colisão é perfeitamente elástica, ou seja, quando ocorre uma colisão, o único efeito é a troca da direção da velocidade.

```
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_)
```

5.3 Detecção de buraco

Função para detectar a localização dos buracos em cada pista.

```
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
```

5.4 Estado Minigolfe

Exibe os objetos de cada fase na tela, atualizando a posição da bolinha.

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

    #apos essas definicoes, temos as definicoes particulares de cada fase, onde a funcao
        ira desenhar cada objeto (lados da pista, buracos e obstaculos) na tela. Abaixo
        temos um if para selecionar a fase que sera desenhada

    if fase == 1:
        fase1()
    elif fase == 2:
        fase2()
    elif fase == 3:
        fase3()
    elif fase == 4:
        fase4()

    #atualizacao da posicao e velocidade da bolinha
    b.move(dt)
    b.xlr8(dt)

    #buscando colisoes da bolinha
    detecta_colisao()
    detecta_buraco()

    #desenhando a bolinha
    fill(255)
    stroke(255)
    ellipse(b.p.x,b.p.y, 2*b.r, 2*b.r)
```

5.5 Definições finais

Definições quanto ao tamanho da tela e comandos que permitem ao usuário realizar as tacadas.

```
def setup():
    size(800, 600)

def draw():
    global estado
    estado()

def mouseDragged():
    global inc, ver

    #condicao para nao aceitar tacadas se a bola estiver se movendo
    if b.v.mag() == 0:
        if sqrt((mouseX - b.p.x)**2 + (mouseY-b.p.y)**2) <= r+10:
            ver = True

        if ver:
            inc.x = b.p.x - mouseX
            inc.y = b.p.y - mouseY
            stroke(255,0,0)
            line(b.p.x,b.p.y,mouseX,mouseY)

def mouseReleased():
    global inc, ver, estado, Minigolf
    if estado == Minigolf:
        if ver:
            b.v.add(inc/300)
            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.