



Introdução à Ciência da Computação

Relatório - Trabalho Final

Apresentação 27/06/2024

INTEGRANTES:

- Carlos Eduardo Cintra Siqueira (15445279)
- Felipe Assis Bernardes Falvo (15636682)

DOCENTE:

- André Smaria

**São Carlos - SP
2024**

1. Biblioteca PYMUNK

A biblioteca PYMUNK, escrita em linguagem C, é principalmente utilizada para simulações de física 2D, em especial para casos onde ocorrem interações entre corpos. Seus principais recursos são:

- Simulação de Objetos:

Antes de criar o corpo em si, é necessário sempre a criação de um espaço de simulação por meio de (`pymunk.Space()`), por onde a física do sistema será estruturada.

Em relação aos objetos, definimos o corpo através do comando (`pymunk.Body()`), em seguida é declarado o formato geométrico do mesmo através de (`pymunk.Circle()`), por exemplo, onde nesse caso é para um círculo, porém existem outros diversos formatos para utilização.

O PYMUNK também oferece a opção de fazer com que esses corpos sejam estáticos, cinemáticos ou dinâmicos, sendo importantíssimo para o nosso projeto, visto que a parede, bola e o jogador devem se comportar de maneiras extremamente diferentes, facilitando na criação da simulação

- Colisões:

Sendo o motivo principal, essa biblioteca possibilita utilizar os conhecimentos adquiridos no curso de Física 1 e aplicá-los em programação, visto que existe a função (`pymunk.elasticity()`), que simula o coeficiente de restituição de colisão entre os objetos, podendo variar de 0 (totalmente plástica) à 1 (totalmente elástica).

Esse recurso possibilitou simplificar a criação do jogo, já que a interação entre o jogador e a bola é descrita de maneira simples e objetiva, possibilitando uma compreensão mais exata do código e maior facilidade na resolução de “bugs” encontrados.

- Detecção de Colisões:

Outro recurso importante do PYMUNK é a detecção de colisões. A biblioteca permite configurar o comportamento quando dois objetos colidem utilizando (`pymunk.CollisionHandler()`), onde é possível definir funções específicas para diferentes situações. No nosso projeto, por exemplo, utilizamos esse recurso para determinar o comportamento da bola ao colidir com as paredes esquerda e direita, reiniciando o jogo nessas ocasiões.

- Alta interação com o PYGAME:

Sabendo que a biblioteca PYGAME é amplamente utilizada para renderização gráfica de simulações e jogos, ela acaba sendo muito usada para a parte visual do programa. Por outro lado, o PYMUNK é responsável por lidar com a física de interação dos corpos. Isso pode ser observado em nosso trabalho, onde toda a parte de inicialização da tela e taxa de quadros do jogo foi gerenciada pelo PYGAME.

2. Código

```
1  import pygame
2  import pymunk
3  import random
4
5  #inicializamos o pygame
6  pygame.init()
7
8  #definimos o número de pixels que nosso "mapa" vai ter
9  mapa = pygame.display.set_mode((1000,600))
10
11 #definimos na objeto "tempo" que vai delimitar o tempo que as coisas acontecem
12 tempo = pygame.time.Clock()
13
14 #criamos um espaço de simulação da física usando o Pymunk
15 espaco = pymunk.Space()
16
17 #taxa de quadros atualizados por segundo
18 fps = 40
19
20 #definiremos os limites das paredes no espaço do pong
21 PEx = 50
22 PDx = 950
23 topo = 25
24 base = 575
25 Mx = 500
26 My = 300
27
```

Nessa primeira parte do código, observamos a configuração do programa, incluindo a importação das bibliotecas utilizadas, a definição da tela por onde o jogo irá funcionar e as principais variáveis que configuram as paredes.

```
30 #criaremos a bola do pong
31 class Bola():
32     def __init__(self):
33         #definimos que a bola será um corpo que o Pymunk vai interpretar
34         self.body = pymunk.Body()
35         #definimos as posições iniciais da bola
36         self.body.position = Mx, My
37         #definimos a velocidade da bola
38         self.body.velocity = 400, -300
39         #definimos uma "forma" para o corpo, para que possa interagir com demais obj.
40         self.shape = pymunk.Circle(self.body, 8)
41         #definimos a densidade do corpo
42         self.shape.density = 1
43         #definimos a elasticidade do corpo (que será 1) para que as colisões sejam
44         #perfeitamente elásticas
45         self.shape.elasticity = 1
46         #adicionamos a nossa bola no espaço
47         espaco.add(self.body, self.shape)
48         #determinamos um tipo de colisão para o formato da bola
49         self.shape.collision_type = 1
50
51     #definiremos o desenho do nosso objeto BOLA
52     def desenho(self):
53         x, y = self.body.position
54         pygame.draw.circle(mapa, (255,255,255), (int(x), int(y)), 8)
55
56     #definimos que quando o jogo for reiniciar, a bola volte a posição e velocidade iniciais
57     def reiniciar(self, X,Y,Z):
58         self.body.position = Mx, My
59         self.body.velocity = 400*random.choice([-1, 1]), -300*random.choice([-1, 1])
60         return False
```

Logo após, foi criado a bola através da classe Bola. Através desta foi declarada três principais funções. A primeira função é responsável pela configuração da bola em si, tal como sua forma, velocidade e tipo de colisão, conforme demonstrado na linha 45. A segunda cria o corpo da bola dentro do espaço definido, juntamente com sua posição, cor e raio. Já na terceira, assegura que quando o jogo for reiniciado, a bola retorne à posição inicial pré-definida junto com as velocidades iniciais. Ademais, graças a biblioteca “RANDOM”, foi possível aleatorizar a direção por onde a bola é lançada, como mostrado na linha 59.

```

61
62 #definiremos o comportamento das paredes no jogo
63 class parede():
64     #colocaremos dois pontos p1 e p2, pois temos 4 paredes e cada uma vai ter
65     #um ponto 1 e ponto 2 diferentes.
66     def __init__(self, p1, p2, collision_number = None):
67         #definimos que as paredes serão corpos estáticos
68         self.body = pymunk.Body(body_type=pymunk.Body.STATIC)
69         #definimos a forma da parede
70         self.shape = pymunk.Segment(self.body, p1, p2, 10)
71         self.shape.elasticity = 1
72         espaco.add(self.body, self.shape)
73         if collision_number:
74             self.shape.collision_type = collision_number
75     def desenho(self):
76         pygame.draw.line(mapa, (255,255,255), self.shape.a, self.shape.b,10)
77

```

Após ser gerado a bola, foi criada a parede do jogo usando a classe “parede”. Através desta classe, foram escritas duas funções principais. A primeira é responsável essencialmente por definir o objeto dentro do espaço, juntamente com a maneira como ela interage com os outros corpos, onde é importante destacar a linha 68, no qual foi setado que o mesmo atuará como um corpo estático.

Além do mais, é importante ressaltar a linha 73 e 74, pois é nela que fomos capazes de diferenciar o tipo de colisão (um número arbitrário) de cada parede mais a frente. A variável “collision_number” também será caracterizada em um momento mais adiante, quando formos definir cada parede, e essa variável vai se tornar o tipo de colisão, que vamos explicar posteriormente.

```

78 #definiremos o comportamento dos jogadores
79 class jogador():
80     #x é apenas a posição em x que o jogador vai ficar
81     def __init__(self, x):
82         #o corpo será "estático" em relação a não adquirir o momento linear da bola
83         #mas queremos que se mova em y, então usamos o tipo de corpo "KINEMATIC"
84         self.body = pymunk.Body(body_type=pymunk.Body.KINEMATIC)
85         self.body.position = x, My
86         self.shape = pymunk.Segment(self.body, [0, -40], [0, 40], 10)
87         self.shape.elasticity = 1
88         espaco.add(self.body, self.shape)
89
90     #Evitar que o corpo ultrapasse as paredes superiores e inferiores
91     def sair(self):
92         #A função local_to_world converte as coordenadas para a biblioteca
93         #A indexação [1] indica que o que queremos é apenas o eixo y
94         p1_y = self.body.local_to_world(self.shape.a)[1]
95         p2_y = self.body.local_to_world(self.shape.b)[1]
96
97         if p1_y < topo:
98             self.body.position = (self.body.position[0], topo + 40)
99         if p2_y > base:
100             self.body.position = (self.body.position[0], base - 40)
101

```

Nessa parte, começamos a configurar o jogador, no qual foi a tarefa mais complexa dentro o código inteiro. Na primeira função, foi configurado o jogador como nos casos anteriores, porém é importante notar, na linha 84, que o mesmo está configurado para ser um corpo cinemático, já que apesar de rebater a bola, ele não deve receber o momento linear da mesma, tendo que apenas se mover no eixo y de acordo com as teclas apertadas. Já a segunda função, dizemos que ela é a responsável por não permitir que o jogador ultrapasse as paredes de cima e de baixo, para isso foi usado as linhas 94 e 95 para conversão dos valores para a biblioteca e da 97 à 100 para delimitar o espaço.

```
102     def desenho(self):
103         #p1 e p2 são os pontos a e b de coordenadas vistas acima, que definem a altura
104         #do desenho do jogador. Queremos que essas posições sejam em relação ao "mundo"
105         #que é o mapa, então precisamos definir de forma mais específica aqui
106         p1 = self.body.local_to_world(self.shape.a)
107         p2 = self.body.local_to_world(self.shape.b)
108         pygame.draw.line(mapa, (255, 255, 255), p1, p2, 10)
109     #Define as velocidades iniciais do jogador
110     def movimento(self, para_cima = True):
111         if para_cima:
112             self.body.velocity = 0, -600
113         else:
114             self.body.velocity = 0, 600
115
116     def parada(self):
117         self.body.velocity = 0, 0
118     #criaremos uma função jogo, que é o nosso jogo :)
```

Ainda falando sobre o jogador, desenhamos-o no espaço e através do código "local_to_world()", as coordenadas do jogador era convertido para as coordenadas da biblioteca, fazendo com que ele seja um corpo móvel. A próxima função apenas define as velocidades iniciais do jogador quando o jogo é iniciado. Após isso, na função parada, serve apenas para declarar as velocidades quando o jogador encontrar a parede de cima e de baixo.

```

119 def jogo():
120     #definimos nosso objeto Bola() no jogo
121     bola = Bola()
122
123     #definimos as quatro paredes
124     parede_esquerda = parede([PEx, topo], [PEx, base], 2)
125     parede_direita = parede([PDx, topo], [PDx, base], 2)
126     parede_topo = parede([PEx, topo], [PDx, topo], 3)
127     parede_base = parede([PEx, base], [PDx, base], 3)
128
129     #definiremos jogador 1 e 2
130     jogador1 = jogador(PEx + 50)
131     jogador2 = jogador(PDx - 50)
132
133     #collision handler basicamente diz pra simulação o que fazer quando dois objetos
134     #colidem
135     ponto = espaco.add_collision_handler(1,2)
136     ponto.begin = bola.reiniciar
137     #se um evento do tipo "QUIT"(no nosso caso) fechar a janela do jogo), o jogo vai
138     # fechar e a função vai retornar o loop.

```

Nessa parte do código definimos uma função geral para o jogo, onde todas as classes irão ser utilizadas e declaradas, juntamente com suas variáveis. Para isso definimos o objeto Bola e as paredes, que tem suas coordenadas e “collision_number” definidos. Como o “collision_number” define o tipo de colisão de cada parede, nas linhas 135 e 136, conseguimos configurar o que acontece quando a bola colide na parede esquerda e direita. Para isso, foi usado o recurso “.add_collision_handler(1,2)”, que determina uma situação entre objetos com tipos de colisão determinados, nesse caso, de reiniciar a bola, onde o 1 representa o tipo de colisão da bola e o 2 representa o tipo de colisão da parede esquerda e direita, como representado acima. Já a linha 136 diz que caso haja a colisão, o jogo deve reiniciar. Nas linhas 130 e 131 definimos as coordenadas iniciais de cada jogador.

```

137     #se um evento do tipo "QUIT"(no nosso caso) fechar a janela do jogo), o jogo vai
138     # fechar e a função vai retornar o loop.
139     while True:
140         for click in pygame.event.get():
141             if click.type == pygame.QUIT:
142                 return
143
144         #define uma ação para caso uma tecla for pressionada
145         teclas = pygame.key.get_pressed()
146         if not jogador2.sair():
147             #tecla SETINHA PRA CIMA faz mover para cima
148             if teclas[pygame.K_UP]:
149                 jogador2.movimento()
150             #tecla SETINHA PRA BAIXO faz mover para baixo
151             elif teclas [pygame.K_DOWN]:
152                 jogador2.movimento(False)
153             else:
154                 jogador2.parada()
155

```

O while acima foi criado especificamente para que quando clicar no botão de fechar do programa ele feche. Já abaixo, foi utilizado novamente o pygame, porém dessa vez para ele está sendo utilizado para a detecção das teclas dos jogadores, onde a setinha de cima e de baixo foram usadas para mover o jogador. No começo da condicional, foi utilizada a condição “if not” para trazer a ideia do jogador não ultrapassar a parede e no final para demonstrar que quando ele chegar na parede ele pare e sua velocidade seja igual a zero

```
156         if not jogador1.sair():
157             if teclas[pygame.K_w]:
158                 #tecla W faz mover para cima
159                 jogador1.movimento()
160             #tecla S faz mover para baixo
161             elif teclas [pygame.K_s]:
162                 jogador1.movimento(False)
163             else:
164                 jogador1.parada()
165
166         #define cor do mapa
167         mapa.fill((0,0,0))
168         #definimos o objeto bola agora com seu desenho circular
169         bola.desenho()
170
171         #definiremos as paredes agora com seu desenho
172         parede_esquerda.desenho()
173         parede_direita.desenho()
174         parede_topo.desenho()
175         parede_base.desenho()
176
```

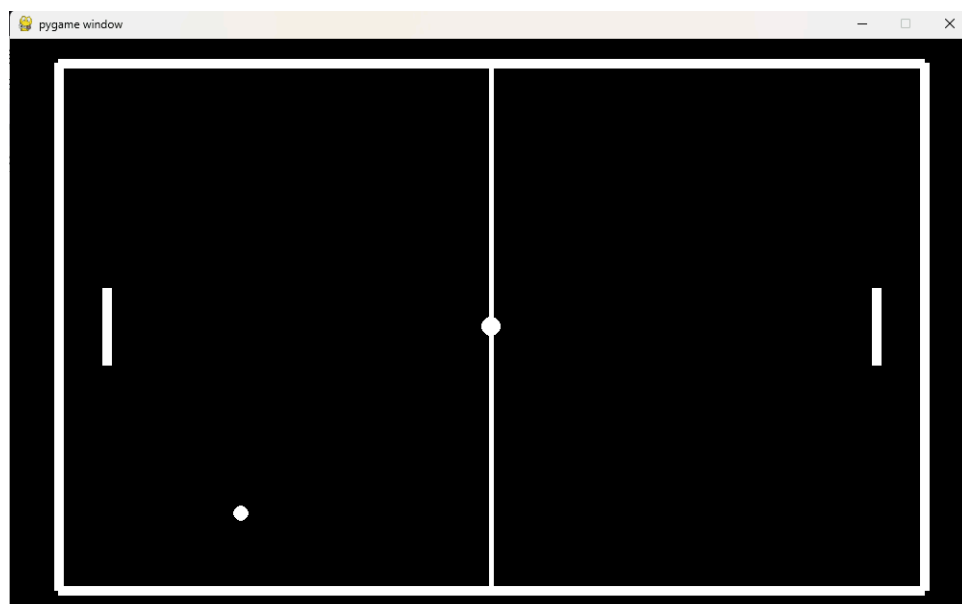
A mesma ideia para o jogador 2 foi utilizada para o jogador 1, evitando ultrapassar a parede e declarando a velocidade da mesma quando encostar no extremo. A linha 167 serve apenas para definir a cor do mapa como sendo sempre preto, para renderização. Na parte final usamos “.desenho” em todos os objetos (paredes, jogadores e a bola) para defini-los no espaço.

Feito isso, com o comando “pygame.draw.line” e “pygame.draw.circle”, com seus devidos parâmetros (mapa, posição e grossura (ou raio)), definimos uma linha e um círculo no meio do mapa para que quem esteja jogando consiga delimitar onde é o meio do mapa.

Com todos os objetos, relações, espaço e mapa definidos, vamos fazer o jogo ser executado. Para isso, ainda dentro do loop do While, usamos o comando “pygame.display.update”, que atualiza a tela numa taxa de frames escolhida, definida pelo comando “tempo.tick(fps)”, e para que o jogo e o espaço físico do PYMUNK seja executado numa mesma taxa de frames, usamos o comando “espaco.step(1/fps)”. Feito tudo isso, temos a função “jogo” pronta. Escrevemos ela e em seguida “pygame.quit” apenas para definir os eventos de “quit”, que no nosso caso seria fechar o programa.

```
177     #definiremos os jogadores com seus devidos desenhos
178     jogador1.desenho()
179     jogador2.desenho()
180
181     pygame.draw.line(mapa,(255,255,255), [Mx, topo], [Mx, base], 5)
182     pygame.draw.circle(mapa, (255,255,255), [Mx, My],10)
183
184     # atualiza a tela durante esse loop para cada frame atualizado
185     pygame.display.update()
186
187     # define o fps máximo do jogo
188     tempo.tick(fps)
189
190     #definimos que o espaço de simulação do Pymunk será atualizado em 40 fps
191     espaco.step(1/fps)
192
193     jogo()
194     pygame.quit()
```

Executando o programa, temos nossa recriação do jogo Pong utilizando o Pygame e o Pymunk, principalmente.



3. Programa Criado

Inicialmente, decidimos criar uma simulação de colisão simples, focando no coeficiente de restituição, que é o principal aspecto desta biblioteca. Entretanto, após pesquisas no Google e no YouTube, encontramos projetos semelhantes utilizando PYMUNK, o que nos mostrou que seria viável desenvolver esse projeto. Dessa forma, como maneira de nos desafiar ainda mais, assistimos vídeos e procuramos em sites informações de como realizar esse jogo tão famoso.

Durante as pesquisas, encontramos um vídeo com um projeto semelhante ao que estávamos planejando. No entanto, ao testá-lo como referência, identificamos alguns bugs. Um exemplo disso é que as colisões com as paredes esquerda e direita não resetavam o jogo, portanto o grupo se reuniu e começou a pesquisar resoluções sobre os problemas existentes.

Para a solução dos problemas de primeira tentávamos entrar no site do PYMUNK e do PYGAME para entendermos as variáveis que cada recurso que eles disponibilizam cobrava, mas se caso após horas pesquisando e não descobrindo o erro, utilizámos o chatGPT para nos explicar quais eram as variáveis que faltavam dentro de certa funcionalidade específica

4. Aprendizado

Com relação ao aprendizado em geral, esta foi uma oportunidade única para expandir nosso conhecimento em Python. Gostaríamos de destacar o uso da Programação Orientada a Objetos. Quando percebemos que o projeto poderia ser desenvolvido utilizando essa técnica, não hesitamos. Apesar de termos encontrado familiaridade com essa forma de organização de código através das aulas e listas oferecidas durante o curso, a chance de aplicá-la em um projeto real nos ajudou ainda mais a compreender esse conteúdo de maneira profunda.

Como foi comentado acima, houve bugs durante a criação do projeto, e apesar de ser algo confuso inicialmente, a oportunidade de ter que descobrir o motivo do erro, nos trouxe ainda mais aprendizados, pois tivemos que ir ainda mais a fundo nas bibliotecas e recursos que estávamos utilizando.

5. Referências


<https://www.pymunk.org/en/latest/>

<https://www.pymunk.org/en/latest/examples.html>

<https://www.pygame.org/wiki/tutorials>

<https://www.pygame.org/news>

 Pong | Pymunk/PyGame Projects

 Learning Pygame by making Pong