Trabalho Prático de Fundamentos de Física

Bruno Rodrigues a21805853

Fábio Pereira a21805862

Índice

Introdução

O objetivo deste trabalho é demonstrar e aplicar as propriedades físicas lecionadas durante este semestre. Para tal iremos detalhar neste relatório todos os processos e formulas físicas utilizadas assim como a nossa abordagem a nível logico e de implementação.

A plataforma utilizada foi o *pygame* visto que se trata de uma plataforma baseada em Python. Aqui achamos que a utilização de Python seria a melhor opção visto que de um ponto de vista de cálculo matemático e aplicações físicas a utilização desta plataforma torna-se extremamente prática na sintaxe e lógica a nível de codificação. Deste modo deixa-nos espaço para nos preocupar-nos com temas como a parte lógica do problema.

Cenário Escolhido

O cenário escolhido trata-se de uma catapulta com um projétil. Aqui o objetivo é demonstrar as características e comportamentos observáveis num movimento balístico ou mais fisicamente falando um movimento obliquo.

Com este cenário pretende-se aplicar os conceitos como funções horarias das posições (horizontal e vertical) e componentes das velocidades assim como noções sobre angulo e lançamento e trajetórias.

Achamos que dentro do que nos foi proposto este seria o cenário ideal visto que permite demonstrar o que é suposto e ao mesmo tempo trata-se de um cenário prático bastante comum e de fácil implementação lógica.

É também um dos fenómenos físicos que mais prevalece no nosso dia-a-dia sendo a sua exposição oportuna e acessível a todos.

Planificação do Trabalho

A planificação do nosso trabalho baseou-se numa estruturação por etapas.

Antes de mais foi necessária uma adaptação à linguagem a nível de sintaxe. Apesar de termos tido contacto com algumas linguagens de programação nunca nos tínhamos deparado com Python. A maior adaptação foi a nível de sintaxe, no entanto, consideramos este obstáculo praticamente inexistente visto que a linguagem apresenta uma sintaxe fácil de interiorizar.

- Posto isto a divisão por etapas foi estruturada da seguinte forma:
- Familiarização a nível de sintaxe, classes e métodos
- Definição do cenário e fenómeno físico a abordar
- Reunião das fórmulas físicas necessárias para a implementação
- Escolha dos *assets* para a nossa simulação
- Codificação da nossa demonstração no *pygame*

Descrição dos Modelos Físicos

Os vetores e respetivas equações de deslocamento e velocidade são as seguintes:

- θ é o ângulo de lançamento
- Velocidade inicial vertical e horizontal, e velocidade instantânea:

$$\sin\theta = \frac{v_{0,y}}{|\vec{v}_0|} \qquad \cos\theta = \frac{v_{0,x}}{|\vec{v}_0|}$$

$$v_{0,y} = |\vec{v}_0|\, sen\,\theta \qquad v_{0,x} = |\vec{v}_0|\cos\theta$$

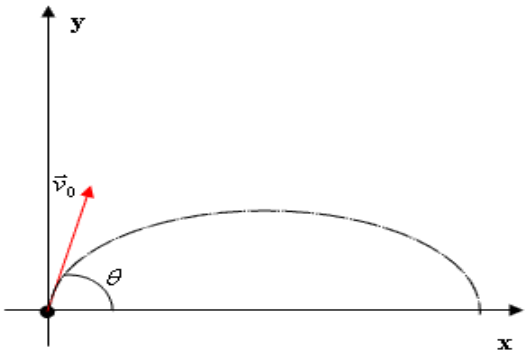
$$v = \sqrt{{v_x}^2 + {v_y}^2}$$

- O espaço s (par-coordenadas x, y), que é dado por:

$$x = x_0 + v_{0,x}\cdot t \qquad x = x_0 + |\vec{v}_0|\cos\theta\cdot t$$

$$y = y_0 + v_{o,y}\cdot t - \frac{1}{2}gt^2 \qquad y = y_0 + |\vec{v}_0|\, sen\,\theta\cdot t - \frac{1}{2}gt^2$$

- Esquematização gráfica do contexto de balística e lançamento oblíquo:



Simulação (Implementação)

A nível de simulação decidimos demonstrar neste relatório a logica por detrás da nossa implementação a nível de código.

Aqui a nossa implementação baseia-se no controle de uma catapulta, mais concretamente o seu braço. Este controle é feito utilizando as teclas direcionais (KEY_UP e KEY_DOWN) sendo estas responsáveis por controlar o angulo de lançamento do nosso projétil.

```
if (keys[pygame.K_UP] or keys[pygame.K_w]) and theta < 45:
    theta = (theta + rot_speed) % 360
    cw_x -= 0.5
    cw_y -= 3
    ball_x +=0.5
    ball_y +=3

if (keys[pygame.K_DOWN] or keys[pygame.K_s]) and theta > 0:
    theta = (theta - rot_speed) % 360
    cw_x += 0.5
    cw_y += 3
    ball_x -=0.5
    ball_y -= 3
```

Acima é possível observar a maneira como as teclas controlam a nossa rotação.

Outro fator que é variável e controlável pelo utilizador é a velocidade de lançamento ou velocidade inicial. Esta é controlada pelas nossas teclas direcionais (KEY_LEFT e KEY_DOWN) e será utilizada para definir as componentes horizontal e vertical da nossa velocidade inicial.

Controle da velocidade inicial:

```
if (keys[pygame.K_LEFT] or keys[pygame.K_a]) and vi > 50:
    vi -= 1
if (keys[pygame.K_RIGHT] or keys[pygame.K_d]) and vi < 200:
    vi += 1
```

Formulas da componente horizontal e vertical da velocidade inicial

```
vx = vi * cos(radians(theta))
vy = vi * sin(radians(theta))
```

$$v_{0_x} = |\overrightarrow{v_0}| \cdot cos\theta \qquad v_{0_y} = |\overrightarrow{v_0}| \cdot sen\theta$$

A definição de a nossa pedra ter ou não sido lançada baseia-se na alternância entre dois estados que definimos. Incluímos uma *flag* inicialmente definida como "*false*" e quando premimos a tecla espaço mudamos o estado dessa *flag* para "*true*". As nossas condições e *loops* baseiam-se em torno desta *flag*.

Como é possível observar abaixo utilizamos a tecla (K_SPACE) para despoletar a mudança de estado para "*true*". Ao definir este estado a "*true*" inicia-se o cálculo das nossas componentes horizontal e vertical da velocidade inicial já referidas acima.

```
if keys[pygame.K_SPACE]:
    s = (100, 500)
    si = s
    t = 0
    travel = True
    # set the initial velocity
    vx = vi * cos(radians(theta))
    vy = vi * sin(radians(theta))
```

Posteriormente vai também ser calculado a cada iteração do nosso *loop* a função horária da nossa posição horizontal e vertical.

```
t += clock.get_time()/1000
s = (si[0] + (vx*t)*2 , si[1] - (((vy*t) + (0.5*G*t*t)))*2) # *2 for pixel-to-meter ratio adjustment
v = sqrt(pow(vx, 2) + pow(vy, 2))
screen.fill((0, 0, 0))
screen.blit(ballImg, s)
```

$$x = x_0 + v_{0_x} t \qquad y = y_0 + v_{o_y} t + \frac{1}{2} g t^2$$

Aqui serão utilizadas as nossas posições x e y do nosso projétil (s[0] e s[1]) e as nossas componentes vx e vy da velocidade definidas inicialmente aquando da mudança de estado da nossa "*flag*".

Aqui medimos também o tempo de voo do nosso projétil utilizando a função *Clock.get_time* que retorna a diferença em milissegundos entre o *frame* atual e o anterior.

Finalmente, para detetarmos o impacto do nosso projétil no solo verificamos quando o nosso s[1] (componente vertical da nossa posição) se encontra igual à componente vertical da nossa catapulta.

Para tal implementamos o seguinte:

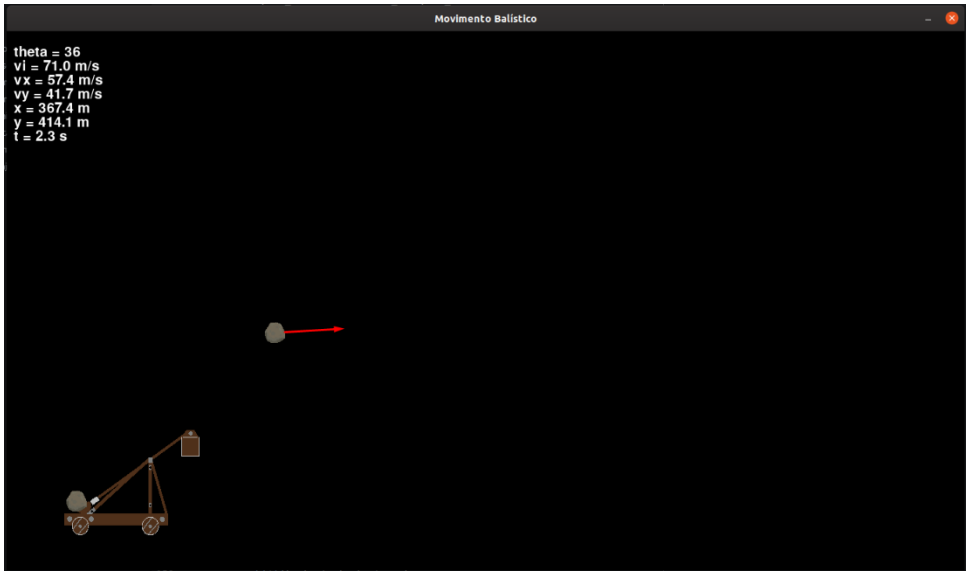
```
if s[1] >= 630:  # ground collision
    travel = False
    vx = 0
    vy = 0
    vi = 5
    s = (100, 500)
    si = s
    t = 0
```

Aqui usamos uma condição que em caso da mesma se verificar definimos a nossa *flag* a "false" novamente e damos "*reset*" a todas as nossas variáveis. De notar que o nosso projétil parte sempre do mesmo ponto visto que resolvemos deixar assim por acharmos que não seria imperativo para motivos de demonstração e para o objetivo deste trabalho.

Testes e Resultados

É interessante verificar a diferença entre simulações com forças e ângulos de lançamento diferentes, bem como reparar na importância da aceleração gravítica.

Para uma constante gravitacional de 20 m/s^2:



Já para G = 10 m/s^2:

Considerações Finais

Com este trabalho foi-nos possível cimentar conceitos relacionados com aspetos físicos abordados na aula.

O estudo de um fenómeno físico para posterior implementação permitiu-nos conhecer mais em detalhe a maneira como se processam estes fenómenos físicos e as relações matemáticas envolvidas.

Acreditamos que de certa forma este trabalho contribui para nos enriquecer também a nível da programação em si visto que a utilização da mesma num paradigma científico é pouco comum no nosso curso.

Assim consideramos que este trabalho foi contributivo para o conhecimento que é suposto obter nesta disciplina e que de certa forma foi adaptado ao nosso curso, algo que devia ser feito em todas as cadeiras.

Para concluir esperamos que o trabalho espelhe o pretendido com o mesmo e que satisfaça as condições impostas pela disciplina.

Repositório de desenvolvimento do projeto:

PythonBallistics em GitHub