Relatório - Modelagem de Fenômeno Físicos

Rener e Jorge

3 de outubro de 2019

Descrição Geral

O objetivo do projeto, foi construir um jogo, implementado no módulo Python do Processing, que mistura a diversão de uma "Guerra de Lançadores de Mísseis"com conceitos de cinemática vetorial. O jogo começa com uma pequena narração dizendo que você (jogador) após compra um lançador de mísseis, deve se proteger de outros lançadores.

O jogador é um representado por um círculo no canto inferior esquerdo da tela. No decorrer do jogo, surgiram do topo da tela, pequenos outros círculos, que são os lançadores inimigos que devem ser combatidos. Ao dar um clique na tela o usuário define uma direção na qual seu míssil deva seguir para conseguir acertar os inimigos, todos os objetos sofrem com a aceleração da gravidade e força de resistência do ar.

Com isso, se a bola principal atingir os limites de tela laterais e inferior, ela retorna a sua posição inicial; No momento em que ela atinge uma bola inimiga, esta desaparece, e a principal retorna novamente para o começo. Todas essas colisões são contatas internamente para atualizar a Fase do Jogo, que quanto mais avançada, maior o nível de dificuldade. Em alguns níveis surge um obstáculo a mais que é um vento de natureza aleatória que pode ajudar ou atrapalhar a jogabilidade.

Conceitos Físicos e Implementação

Lançamento de Projétil com resistência do ar

O primeiro conceito abordado é a modelagem de lançamento de projéteis considerando a resistência do ar, que será a modelagem da bola principal. Este modelo sugere que as componentes do vetor da Força de Resistência do Ar ($\vec{F}$) são proporcionais à velocidade do objeto. Pela Segunda Lei de Newton, o que temos é:

$$\vec{F_r} = m \cdot \vec{a} = k\vec{v}$$

Ou seja, $\vec{a} = \frac{k}{m}\vec{v}$

A constante k é dada por $-\frac{1}{2}\rho A C_r$ onde:

- ρ é a densidade do fluido, que no caso é o ar, que para simplificar, arredondamos para 1;
- A é a área de contato do objeto com o fluido. No caso da nossa bolinha, considerando que ela representa uma esfera de mesmo raio no $\mathbb{R}^3$, está área de contato será sempre metade da área total da superfície, ou seja $A = \frac{4\pi r^2}{2} = 2\pi r^2$;
- C_r é o Coeficiente de resistência, que se assemelha ao coeficiente de atrito e seu valor depende do formato do objeto. No caso de uma esfera, $C_r = 0,47$;

O código onde essas variáveis são declaradas está logo abaixo, e foi aplicado um fator de conversão para pixels na escala de 1 cm = 1 pixel.

```
    rho = 1 #
    coef_ball = 0.47
    A = [2*PI*r*r/10000, 2*PI*bomb_r*bomb_r/10000]
    ball_k = rho*coef_ball*A[0]/2
```

A variável A, também armazena a contante k das bombinhas que caem verticalmente.

O Vento e a velocidade relativa

Em certos níveis de dificuldade do jogo, surge um vento horizontal de sentido arbitrário que pode atrapalhar ou ajudar o usuário. A sua velocidade v_v é dada aleatoriamente nestes níveis.

Para generalizar a modelagem e considerar o impacto que o vento gerará, mudasse apenas uma variável do cálculo da força de retardo. Teremos agora a velocidade relativa,

$$\vec{a_r} = \frac{k}{m}\vec{v_r}$$

,

onde $\vec{v_r} = v_v - \vec{v}$

OBS.: No código os cálculos de cada componente desses vetores é feito separadamente.

Como inicialmente $v_v = 0$, o que ocorre neste caso, é que a fórmula recai em $\vec{a_r} = -\frac{k}{m}\vec{v}$

Se a velocidade do vento estiver no mesmo sentido que o movimento, isso reduzirá a força de resistência do ar, ajudando a impulsionar o objeto. Caso contrário, a resistência do ar será maior, o que retardará ainda mais a bolinha.

Aplicação do Método de Euler

Para atualizar a posição e aceleração da bola principal, utilizamos o Método de Euler, pois o que temos é aceleração em função da velocidade, além de acelerações e velocidades iniciais. É um problema de equações diferenciais de valor inicial, que optamos por usar a resolução numérica por conta do ciclo do jogo ser rápido.

A velocidade é incrementada a cada frame da seguinte forma:

$\vec{v}_{r(n+1)} = \vec{v}_{r(n)} + \Delta t \cdot \vec{v}\,'_{r(n)}$

$= \vec{v}_{r(n)} + \Delta t \cdot \vec{a}_n$

Onde $\vec{v}_{r(n+1)}$ é a velocidade relativa em dado instante, $\vec{v}_{r(n)}$ é a velocidade relativa no instante anterior. De forma geral o índice $n+1$ representa o valor da variável atual, e n o seu valor no instante anterior.

Δt é um incremento que deve ser um número razoavelmente pequeno que se aproxima de 1 frame.

Após o incremento da velocidade, a aceleração é recalculada da seguinte forma, utilizando as definições de força de resistência e aceleração inicial $\vec{a}_0$:

$$\vec{a}_{n+1} = \frac{k}{m} \cdot \vec{v}_{r(n+1)} + \vec{a}_0$$

Note que a aceleração não usa o Método de Euler diretamente, ela simplesmente aplica a fórmula já deduzida, de resistência do ar, que deve ser reatualizada para garantirmos velocidade variável.

A posição é incrementada pela seguinte fórmula:

$$p_{n+1} = p_n + v_{r(n)}\Delta t + \frac{a_{n+1}\Delta t^2}{2}$$

Note que a apesar de não estarmos tratando de movimento uniformemente variável, a fórmula da posição é basicamente a mesma do M.U.V. Isso pois no pequeno intervalo de Δt podemos por aproximação dizer que o modelo é uniformemente variado, pois neste intervalo mínimo a aceleração é constante.

No caso da bola principal, temos dois blocos de códigos que definem valores iniciais: os valores iniciais de repouso, e os valores iniciais no caso de clique do mouse:

```
    ball = 0
    m = 1 #
    r = 20 #
    wind_speed = PVector(0, 0) #
    ball_speed = PVector(0, 0)
    gravity = 9.8 #
    acceleration = PVector(0, gravity)

    bomb_list = []
    bomb_acceleration = PVector(0, gravity)
```

```
    bomb_r = 40 #
    points = 0
    count = 0
    perdeu = 0

    def setup():
        global ball, r

        size(900, 900) #
        ball = PVector(r, height - r)
```

O bloco do clique do mouse, como dissemos, define uma $\vec{v}_0$ na direção do clique, e aciona a aceleração da gravidade.

```
    def mouseClicked():
    global ball, ball_speed, acceleration, t
    if ball.y == height - r:
        ball_speed = (PVector(mouseX, mouseY) - ball)/3 #
        acceleration.y = gravity
```

Bibliografia

Khan Academy. *"Air and Fluid Resistance."*. 7 Set. 2018. Acesso em 2 Out. 2019 Deisponível em https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-forces/a/air-and-fluid-resistance