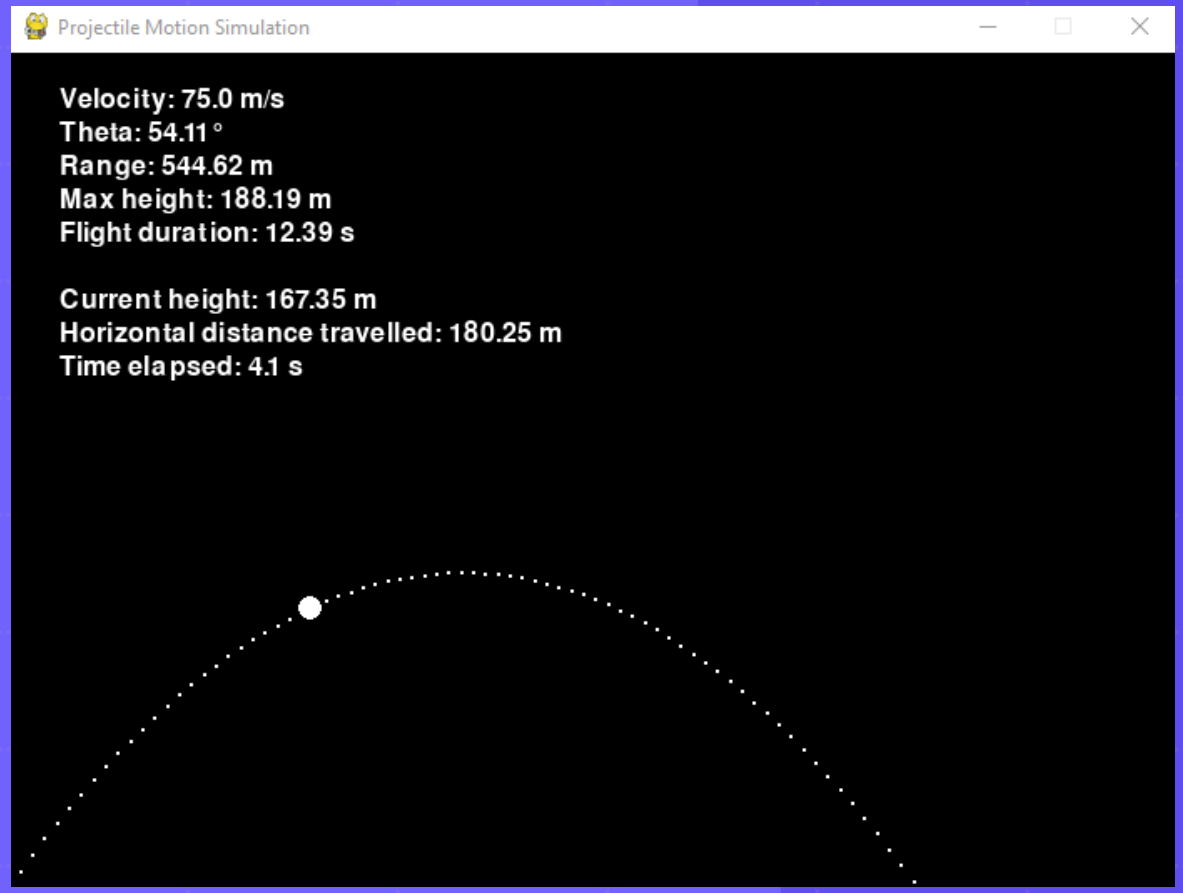
Simulazione grafica del moto di un proiettile con Python

Rover Simone Matr. 933288 - 27/2/2023

Obiettivo del progetto

- Si vuole fornire una rappresentazione realistica.
- I dati e la simulazione devono essere quanto più precisi possibili.
- L'esperienza può essere personalizzata scegliendo l'angolo di lancio del proiettile.
- La velocità del corpo è fissata a 75 $\frac{m}{s}$



Le formule

Vengono calcolati per ogni lancio le informazioni principali, quali:

- **Tempo di volo**: $t_{volo} = 2\frac{v \sin\theta}{g}$
- **Altezza massima**: $h_{max} = \frac{(v \sin\theta)^2}{2g}$
- **Portata del lancio**: $x_f = \frac{v^2 \sin 2\theta}{g}$

Dato che la velocità è fissata, l'unica variabile è l'angolo θ.

Implementazione

Per la visualizzazione e gestione degli eventi utillizzo la libreria **Pygame**.

Ad ogni click del mouse, se non è già in corso una simulazione, prendo l'angolo descritto dal cursore e la base della finestra per impostare l'angolo della simulazione.

Aggiorno le informazioni nel run loop finché l'angolo è impostato e non è stato raggiunto il suolo; se viene premuto ancora il mouse, pulisco lo schermo in preparazione per la prossima simulazione, se invece viene raggiunto il suolo, arresto la simulazione.

Occorre impostare dati iniziali quali il numero di fotogrammi al secondo. Sarà cruciale per stabilire l'intervallo di integrazione numerica. Infatti, se consideriamo il fotogramma la nostra unità temporale, aggiornando il display 60 volte al secondo, la nostra nuova unità temporale sarà 1/60, ovvero 0.0166… s.

Assegnando 0.0166… allo step di integrazione otteniamo la simulazione più realistica possibile.

Codice – Gestione evento Mouse

```
# Check for mouse click event
elif event.type == pygame.MOUSEBUTTONDOWN:
    if not angle_set:
        screen.fill(BLACK)
        pygame.display.update()
        # Get the angle of projection from the mouse click
        theta = -math.atan2(event.pos[1] - y, event.pos[0] - x)
        vx = v0 * math.cos(theta)
        vy = v0 * math.sin(theta)
        # Calc data
        horizRange = (v0**2 * math.sin(2*theta)) / g
        maxHeight = (vy**2) / (2*g)
        flightTime = 2*(vy/g)
        # Update displayed infos
        angleText = font.render("Theta: {}°".format( round((theta*180)/math.pi, 2) ), True, WHITE)
        rangeText = font.render("Range: {} m".format( round(horizRange, 2) ), True, WHITE)
        heightText = font.render("Max height: {} m".format( round(maxHeight, 2) ), True, WHITE)
        timeText = font.render("Flight duration: {} s".format( round(flightTime, 2) ), True, WHITE)

    else:
        x = 0
        y = height
        t = 0
        trajectory = []
        screen.fill(BLACK)
        screen.blit(initPrompt, (30,20))
        angle_set = False
        ground_reached = False
```

Codice – Integrazione numerica

```
# Update the position and velocity of the projectile
x += vx * dt
y -= (vy * dt + 0.5 * g * dt ** 2)
vy -= g * dt
t += dt
# Check ground reached
if (height - y <= 0):
    ground_reached = True
    y = height
    x = horizRange
    t = flightTime

# Update dynamic infos
currentHeightText = font.render("Current height: {} m".format( round(height - y, 2) ), True, WHITE)
currentDistanceText = font.render("Horizontal distance travelled: {} m".format( round(x, 2) ), True, WHITE)
timeElapsedText = font.render("Time elapsed: {} s".format( round(t, 2) ), True, WHITE)

# Draw the projectile on the screen
pygame.draw.circle(screen, WHITE, (int(x), int(y)), 7)
```

Calcolo della traiettoria

La traiettoria è un insieme di coordinate, ovvero due numeri interi.

Viene calcolata in anticipo rispetto alla simulazione, subito dopo che l'utente stabilisce l'angolo del lancio.

Una volta calcolata e disegnata la prima volta, occorre riscriverla ad ogni refresh del display. Per motivi estetici ho deciso di disegnare solo un punto ogni dieci.

- $x(t) = v \cos\theta \, t$
- $y(t) = v \sin\theta \, t - \frac{g}{2} t^2$

Codice – Calcolo della traiettoria

```
# Compute trajectory
# x(t) = vx * t
# y(t) = (vy * t) - (g/2 * t^2)
while not ground_reached:
    trajectory.append((vx*t, height - (vy*t - g/2*t**2)))
    pygame.draw.circle(screen, WHITE, (vx*t, height - (vy*t - g/2*t**2)), 1)
    pygame.display.update()
    time.sleep(0.001)
    t += dt
    ground_reached = t >= flightTime
```

```
# Draw trajectory
for i in range(0, len(trajectory), 10):
    pygame.draw.circle(screen, WHITE, trajectory[i], 1)
```

Conclusione

E’ possibile vedere la simulazione tramite un video su YouTube all’indirizzo https://www.youtube.com/watch?v=kglT-1igxMk .

E’ inoltre possibile visionare e scaricare il codice Python completo nella repository GitHub al seguente indirizzo: https://github.com/SimoneRover/PythonProjectileSim .