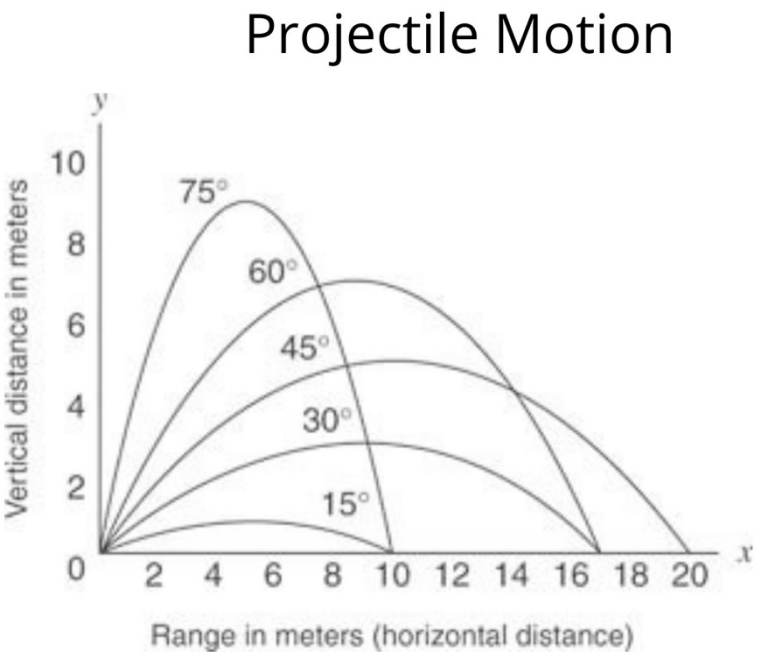
Computer Animation Lab: 08

Contents

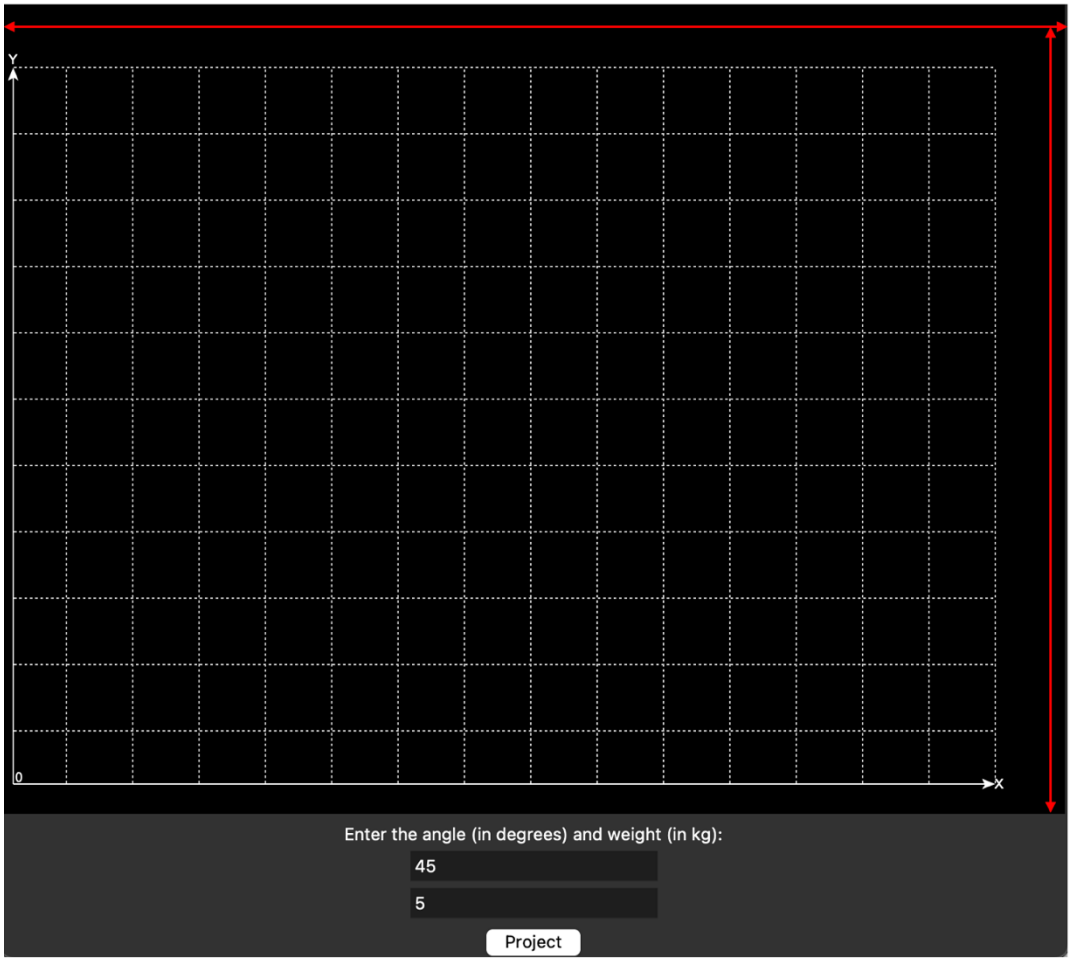
Projectiles

- Projectile refers to an object that is in flight after being thrown or projected.
- In a projectile motion, the only acceleration acting is in the vertical direction which is acceleration due to gravity (g).
- Equations of motion, therefore, can be applied separately in X-axis and Y-axis to find the unknown parameters.
- Components of velocity at time t with initial velocity v_0 and angle θ:
  - Horizontal velocity: $v_x = v_0 \cos(\theta)$
  - Vertical velocity: $v_y = v_0 \sin(\theta) - gt$
- Position at time t:
  - $x = v_0 \cos(\theta_0)\, t$
  - $y = v_0 \sin(\theta_0)\, t - 1/2gt^2$



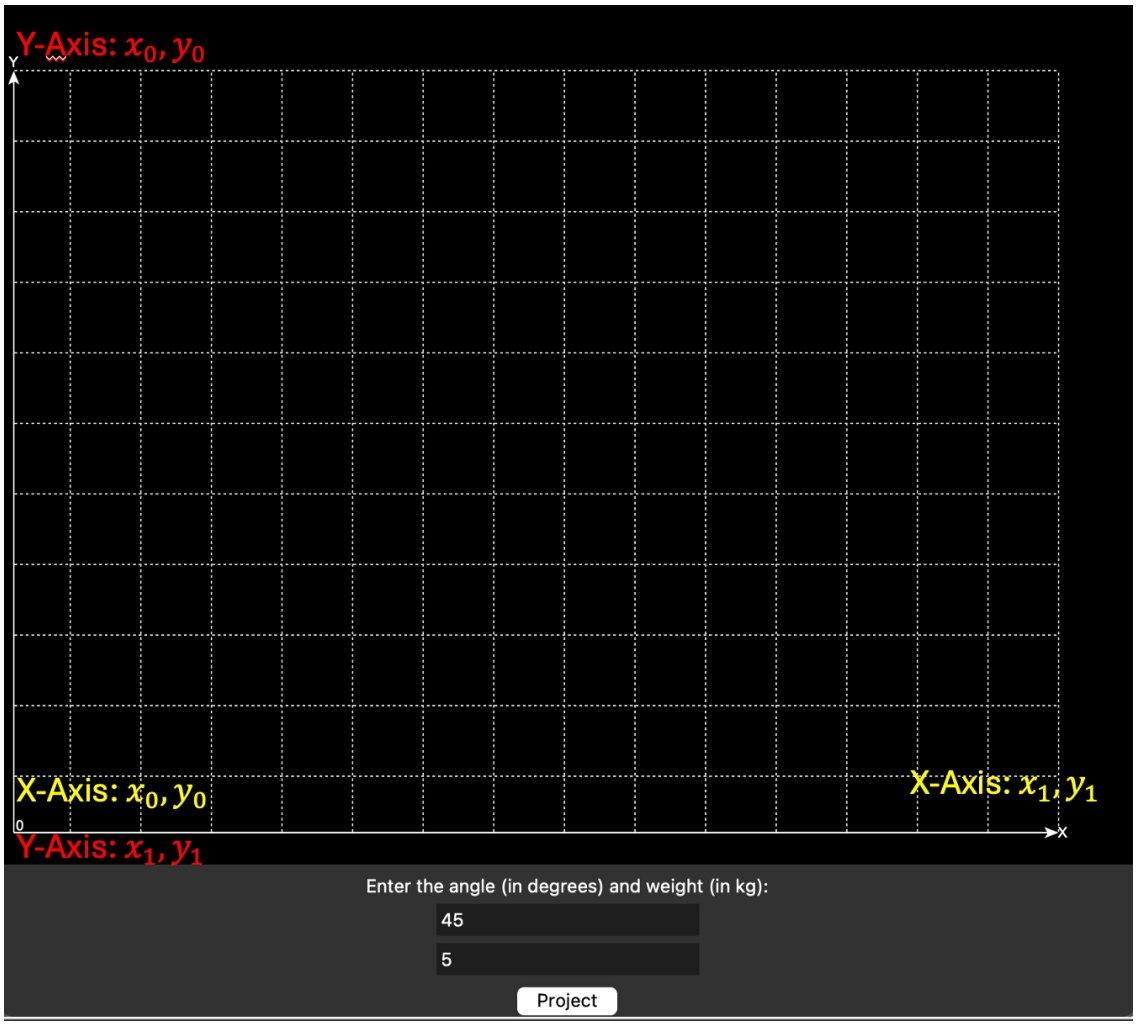
Preparing the Scene

- Canvas:
  - CANVAS_WIDTH = 800
  - CANVAS_HEIGHT = 610

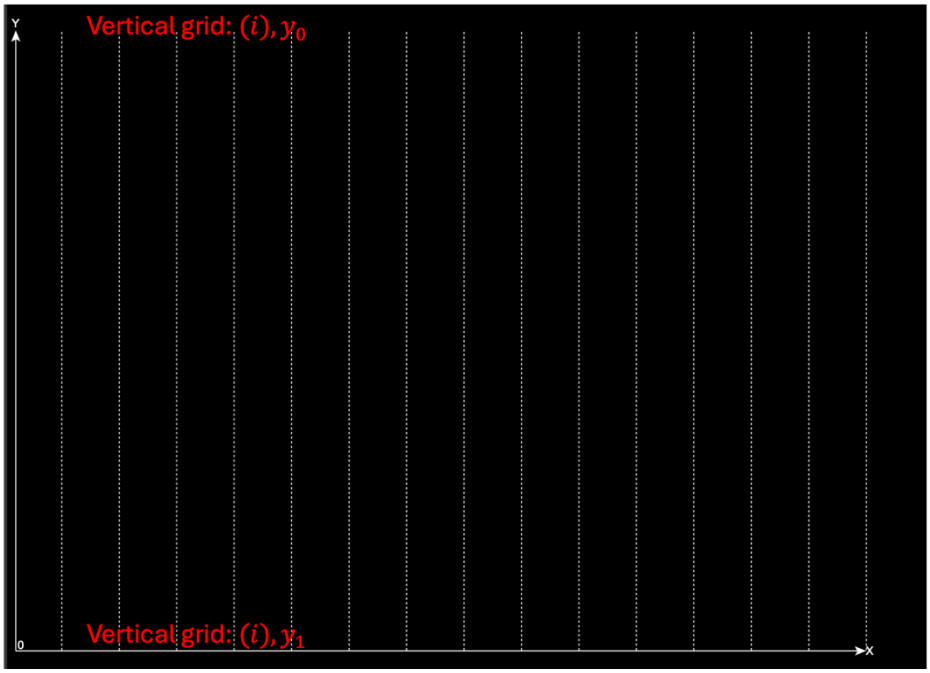


- Axes:
    - X_AXIS_x0 = 10
    - X_AXIS_y0 = 590
    - X_AXIS_x1 = 750
    - X_AXIS_y1 = 590

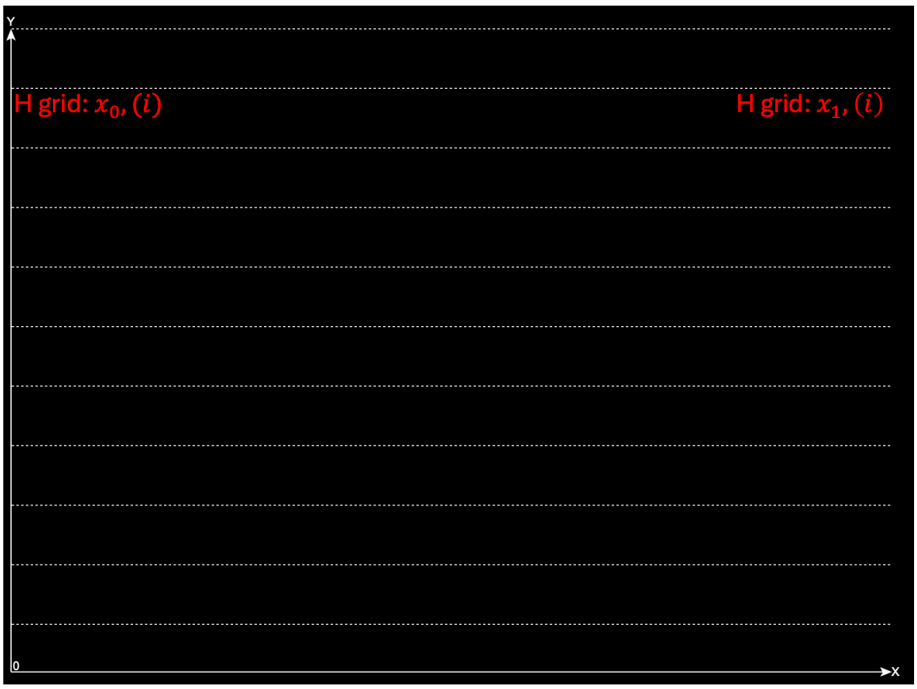
    - Y_AXIS_x0 = 10
    - Y_AXIS_y0 = 590
    - Y_AXIS_x1 = 10
    - Y_AXIS_y1 = 50



- Vertical grid:
  - V_GRID_y0 = 590
  - V_GRID_y1 = 50

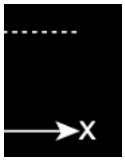


- Horizontal grid
  - H_GRID_x0 = 10
  - H_GRID_x1 = 750

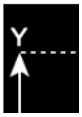


- Origin label:
  - ORIGIN_LBL_x = 12
  - ORIGIN_LBL_y = 585

- X-axis label:
  - X_AXIS_LBL_x = 750
  - X_AXIS_LBL_y = 590



- Y-axis label:
  - Y_AXIS_LBL_x = 10
  - Y_AXIS_LBL_y = 50



- Angle label:
  - ANGLE_LBL_x = 10
  - ANGLE_LBL_y = 600

- Motion constants:
  - `VELOCITY_K = 260.` Used to compute the initial velocity based on the weight.
  - `TIME_STEP = 0.2.` Time to perform the motion for the simulation.
  - `H_VELOCITY_K = 2` Used to add extra velocity in horizontal direction for simulation purposes.
  - `RADIUS_SCALE_FACTOR = 5` Used to draw bigger circles for heavier ones.
  - `EDGE_OFFSET_X = 50` Offset to move away from the x-axis.
  - `EDGE_OFFSET_Y = 550` Offset to move away from the y-axis.
  - `GROUND_Y = 580` The ground is at the line of the x-axis.
  - `GRAVITY = 9.81` The acceleration, which is the gravity.

Creating the Main GUI

```
class ProjectileSimulation:
    def __init__(self, root):
        self.root = root
        self.root.title("Projectile Motion Simulation")

        # Create a canvas to draw the projectile and axes
        self.canvas = Canvas(self.root, width=CANVAS_WIDTH, height=CANVAS_HEIGHT, bg="black")
        self.canvas.pack()

        # Add instruction labels
        self.instruction_lbl = Label(self.root, text="Enter the angle (in degrees) and weight (in kg):")
        self.instruction_lbl.pack()

        # Input fields for angle and weight
        self.angle_entry = Entry(self.root)
        self.angle_entry.pack()
        self.angle_entry.insert(0, "45")

        self.weight_entry = Entry(self.root)
        self.weight_entry.pack()
        self.weight_entry.insert(0, "5")

        # Button to start the simulation
        self.start_button = Button(self.root, text="Project", command=self.simulate)
        self.start_button.pack()

        # Default values
        self.angle = 0  # angle in degrees
        self.weight = 0  # weight of the projectile in kg

        # Physics constants
        self.g = GRAVITY  # acceleration due to gravity (m/s^2)
        self.draw_axes()
```

Draw the Axes

```
def draw_axes(self):
    # Draw the X and Y axes
    self.canvas.create_line(X_AXIS_x0, X_AXIS_y0, X_AXIS_x1, X_AXIS_y1, arrow=LAST, fill="white")  # X-axis
    self.canvas.create_line(Y_AXIS_x0, Y_AXIS_y0, Y_AXIS_x1, Y_AXIS_y1, arrow=LAST, fill="white")  # Y-axis

    # Draw grid lines
    for i in range(50, 800, 50):  # Vertical grid lines
        self.canvas.create_line(i, V_GRID_y0, i, V_GRID_y1, fill="lightgray", dash=(2, 2))
    for i in range(50, 600, 50):  # Horizontal grid lines
        self.canvas.create_line(H_GRID_x0, i, H_GRID_x1, i, fill="lightgray", dash=(2, 2))

    # Label the axes
    self.canvas.create_text(ORIGIN_LBL_x, ORIGIN_LBL_y, text="0", anchor=W, font=("Arial", 10),
                            fill="white")  # Origin label
    self.canvas.create_text(X_AXIS_LBL_x, X_AXIS_LBL_y, text="X", anchor=W, font=("Arial", 10),
                            fill="white")  # X-axis label
    self.canvas.create_text(Y_AXIS_LBL_x, Y_AXIS_LBL_y, text="Y", anchor=S, font=("Arial", 10),
                            fill="white")  # Y-axis label
```

Simulation

```
def simulate(self):
    # Read user inputs
    try:
        self.angle = float(self.angle_entry.get())  # in degrees
        self.weight = float(self.weight_entry.get())  # in kg
    except ValueError:
        print("Invalid input! Please enter valid numbers.")
        return

    # Convert angle to radians
    self.angle_rad = math.radians(self.angle)

    # Calculate initial velocity based on the weight
    self.v0 = VELOCITY_K / self.weight

    # Set initial conditions
    self.time_step = TIME_STEP  # time step for the simulation (s)
    self.time = 0  # start time
    self.x = ORIGIN_LBL_x  # starting x position (origin)
    self.y = ORIGIN_LBL_y  # starting y position (origin), bottom of the canvas
    self.vx = self.v0 * math.cos(self.angle_rad) + H_VELOCITY_K  # horizontal velocity (m/s)
    self.vy = self.v0 * math.sin(self.angle_rad)  # vertical velocity (m/s)

    # Draw the axes and the angle label at the bottom left
    self.draw_axes()
    try:
        self.canvas.delete("angle")
    except:
        pass
    self.canvas.create_text(ANGLE_LBL_x, ANGLE_LBL_y, text=f"Angle: {self.angle}°",
                            anchor=W, font=("Arial", 12), tags="angle", fill="white")

    fill_color = '#%02x%02x%02x' % (randrange(256), randrange(256), randrange(256))
    # Draw the projectile as a circle
    self.radius = RADIUS_SCALE_FACTOR * math.sqrt(self.weight)
    self.projectile = self.canvas.create_oval(self.x - self.radius, self.y - self.radius,
                                              self.x + self.radius, self.y + self.radius, fill=fill_color)

    # Start the animation loop
    self.animate()
```

Run the Animation

```
def animate(self):
    # Update the time
    self.time += self.time_step

    # Calculate the new positions
    ## x(t) = vx * t
    self.x = EDGE_OFFSET_X + self.vx * self.time  # offset to move away from the edge

    ## y(t) = vy * t - 1/2 * gravity * t^2
    self.y = EDGE_OFFSET_Y - (self.vy * self.time - 0.5 * self.g * self.time ** 2)

    # Check if the projectile has hit the ground (close to the bottom of the canvas)
    if self.y >= GROUND_Y:
        self.y = GROUND_Y
        self.canvas.coords(self.projectile, self.x - self.radius, self.y - self.radius,
                           self.x + self.radius, self.y + self.radius)
        return  # End the animation when the projectile hits the ground

    # Move the projectile on the canvas
    self.canvas.coords(self.projectile, self.x - self.radius, self.y - self.radius,
                       self.x + self.radius, self.y + self.radius)

    # Continue the animation
    self.root.after(20, self.animate)
```