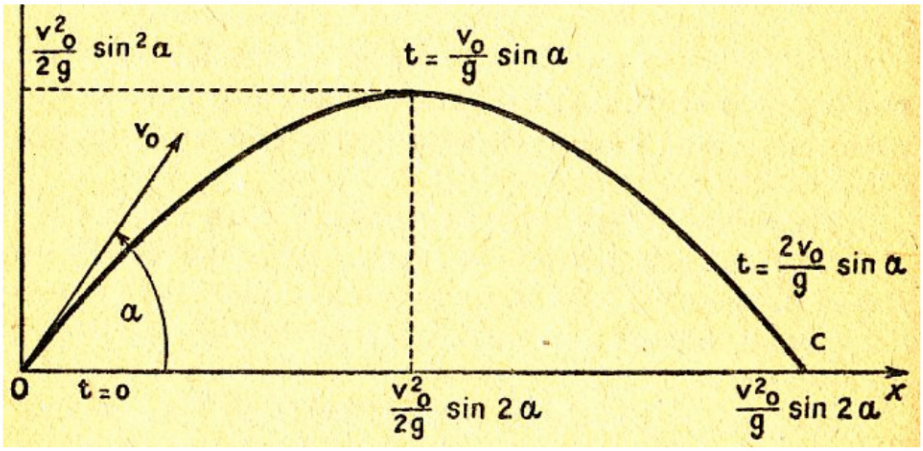
Multimédia et interaction humain-machine

ED animation & simulation

On souhaite réaliser un jeu de tir très simple dans lequel le but est d'atteindre une cible avec un projectile tiré dans le vide (i.e. ne subissant que la pesanteur terrestre).

Le point duquel est tiré le projectile est aléatoire tout comme la position de la cible. On s'assurera tout de même que le projectile est tiré de la gauche de l'écran et que la cible se trouve à droite.



$$\sum \vec{F} = \begin{pmatrix} 0 \\ -mg \\ 0 \end{pmatrix} = m\vec{a} = \begin{pmatrix} m\ddot{x} \\ m\ddot{y} \\ m\ddot{z} \end{pmatrix} \Rightarrow \begin{cases} \dot{x} = v_0 \cos\alpha \\ \dot{y} = v_0 \sin\alpha - gt \\ \dot{z} = 0 \end{cases} \Rightarrow \begin{cases} x = v_0 t \cos\alpha \\ y = v_0 t \sin\alpha - \frac{1}{2} g t^2 \\ z = 0 \end{cases}$$

avec g = 9.81 m/s^2

Question 1 : D'après les équations paramétriques d'un tir de projectile dans le vide (ci-dessus), de quelles valeurs aura-t-on besoin pour calculer la position du projectile ? Quelles variables doivent être déclarées et à quel moment doit-on les initialiser ?

Corrigé :

Nous aurons besoin d'une série de variable pour stocker les conditions initiales de la simulation :

- **la position initiale du projectile,**
- **la vitesse initiale du projectile,**
- **l'angle du jet par rapport à l'horizontale,.**

A cela s'ajoute la position de la cible et le temps en secondes écoulé depuis le début de la simulation.
Ce temps écoulé se calcule en retranchant le temps actuel en secondes (millis()/1000) au temps de début de simulation que nous aurons sauvegardé.

Question 2 : Quels aspects doit-on gérer dans la méthode draw ?

Corrigé :

Dans la méthode draw() il y aura 2 aspects concurrents à gérer :

- **la modification des conditions initiales de la simulation avant de tirer, ceci afin de permettre au joueur de modifier l'angle et la force du tir,**
- **le rendu des différentes étapes de la simulation.**

Il nous faudra donc une variable booléenne permettant de savoir dans quel mode on se trouve.

Question 3 : Donner le code processing permettant de donner visuellement les paramètres de tir et d'effectuer la simulation de tir.

```
Vector2D init, current, target;

int v0 = 100;           // en pixels
float angle = 0;        // en radians

boolean fire = false;   // true quand on est en mode simulation
int startTime;          // date de début de simulation

void setup() {
  size(600, 400);

  initialisation();

  frameRate(50);
}

void initialisation() {
  init = new Vector2D(20+(int)random(50), 10+(int)random(height-50));
  current = new Vector2D(init);

  target = new Vector2D(width-20-(int)random(50), 10+(int)random(height-
50));

  fire = false;
}

void draw() {
  background(128);

  // Dessine le sol
  fill(0);
  line(0, height-10, width, height-10);

  // dessine la cible
  fill(255, 0, 0);
  rect(target.x-5, target.y-5, 10, 10);

  // Dessine direction + force du tir
  strokeWeight(3);
  fill(0, 0, 255);
  line(init.x, init.y, init.x+v0*cos(angle), init.y+v0*sin(angle));
  strokeWeight(1);
```

```
  // Dessine le point init
  fill(0, 255, 0);
  rect(init.x-5, init.y-5, 10, 10);

  // Dessine le projectile
  if (fire)
  {
    float deltaTime = (float)(millis()-startTime)/1000;

    System.out.println(deltaTime);
    current.x = init.x+(int)(v0*deltaTime*cos(angle));
    current.y = init.y+(int)(v0*deltaTime*sin(angle) +
                0.5*9.8*deltaTime*deltaTime);

    if (current.x > width || current.y > height)
      fire = false;

    fill(0, 0, 255);
    strokeWeight(3);
    ellipse(current.x, current.y, 2, 2);
    strokeWeight(1);

    // boom ?
    if (current.x >= target.x-5 && current.x <= target.x+5 && current.y >=
target.y-5 && current.y <= target.y+5)
      initialisation();
  }
}

void keyPressed() {
  switch(key)
  {
    case 'z':
      if (!fire)
        angle -= 0.01;
      break;

    case 's':
      if (!fire)
        angle += 0.01;
      break;

    case 'q':
      if (!fire && v0 > 50) v0--;
      break;

    case 'd':
      if (!fire && v0 < 150) v0++;
      break;

    case ' ':
      fire = true;
      startTime = millis();
      break;
  }
}

public class Vector2D
{
  int x, y;
```

```
  Vector2D(int x, int y)
  {
    this.x = x;
    this.y = y;
  }

  Vector2D(Vector2D v)
  {
    this.x = v.x;
    this.y = v.y;
  }
}
```

Question 4 : Comment gérer le rebond du projectile sur le sol ?

Corrigé :

Pour les rebonds, on peut appliquer le même principe que la loi de lambert pour la lumière : l'angle entre le trajet après rebond et la normale au sol est le même que l'angle entre la normale et le trajet avant rebond.
Il faut donc calculer la normale au point d'impact lors du rebond (facile si le sol est plan) et définir l'angle d'incidence en prenant en les 2 dernières positions du projectile.
Pour obtenir des rebonds plus réalistes, on pourra réduire la vélocité après rebond pour simuler les frottements avec le sol.