BAKURETSU

23 de maig de 2019

1 Pre-Explosion

- P = momentum (initial i or final f)
- m = mass
- V = velocity
- C = particles

Explosion momentum:

$$P_i = P_f \tag{1}$$

$$\forall\ i\ u\epsilon C\ m_u V_u = m_i V_i \tag{2}$$

$$\sum_i^n m_i V_i = P_f \tag{3}$$

Explosion angle:

- E = explosion center
- C = particle center
- φ = angle z-y
- θ = angle x-y

$$r = \sqrt{(E.y - C.y)^2 + (E.x - C.x)^2 + (E.z - C.z)^2}$$

$$\varphi = \arccos \frac{z}{r} \tag{4}$$

$$\theta = \arctan \frac{y}{x} \tag{5}$$

2 Post-Explosion

Parabolic fragment movement:

$$z = z_0 + v_0 t sin\varphi \tag{6}$$

$$x = x_0 + v_0 t cos\theta \tag{7}$$

$$y = -\frac{gt^2}{2} + y_0 + v_0 t sin\theta cos\varphi \tag{8}$$

Collision Detection: Circle vs Circle (or AABB)
Collision Resolution: 3º law of newton

Listing 1: 2D example

```
void Shape::resolveCollision(Shape& A, Shape& B)
{
    // Calculate relative velocity
    sf::Vector2f rv = B.velocity - A.velocity;
    sf::Vector2f n  = Shape::calculateNormal(A, B);
    // Calculate relative velocity in terms of the normal direction
    float velAlongNormal = rv.x * n.x + rv.y * n.y;

    // Do not resolve if velocities are separating
    if (velAlongNormal > 0) return;

    // Calculate restitution
    float e = (A.material.restitution < B.material.restitution) ?
              A.material.restitution : B.material.restitution;

    // Calculate impulse scalar
    float j = -(1 + e) * velAlongNormal;
    j /= A.massData.invMass + B.massData.invMass;

    // Apply impulse
    sf::Vector2f impulse = j * n;
    A.velocity -= A.massData.invMass * impulse;
    B.velocity += B.massData.invMass * impulse;
}
```