**Reflection Documentation**

Below I will explain how the implementation of the VectorHelper class was used to achieve the desired effect, as well as an explanation and formulae for the numerous vectors used for calculating the reflection.

1. Three vectors **topRight**, **bottomLeft**, and **topLeft** are calculated using the **GetAABB()** function of the **other** object. These vectors represent the positions of the corners of the bounding box of the object the grenade collides with.
2. The code then checks the direction of collision by comparing the absolute values of **depth.x** and **depth.y**. This helps determine whether the collision occurs more in the x-direction or the y-direction.
3. If the absolute value of **depth.x** is less than the absolute value of **depth.y**, it means the collision is primarily in the x-direction. In this case, the following steps are executed:
   1. The grenade's **newPosition** is updated by adding **depth.x** multiplied by 2.0f. This moves the grenade in the x-direction to resolve the collision.
   2. The collision normal is calculated by subtracting **bottomLeft** from **topLeft** and passing it to **VectorHelper::GetNormal** to obtain the normal vector.
   3. The velocity of the grenade is updated by calling **VectorHelper::GetReflection** with the current velocity and the normalized collision normal as arguments. This calculates the reflection of the velocity based on the collision normal.
4. If the absolute value of **depth.x** is not less than the absolute value of **depth.y**, it means the collision is primarily in the y-direction. In this case, the following steps are executed:
   1. The grenade's **newPosition** is updated by adding **depth.y** multiplied by 2.0f. This moves the grenade in the y-direction to resolve the collision.
   2. The collision normal is calculated by subtracting **topRight** from **topLeft** and passing it to **VectorHelper::GetNormal** to obtain the normal vector.
   3. The velocity of the grenade is updated by calling **VectorHelper::GetReflection** with the current velocity and the normalized collision normal as arguments. This calculates the reflection of the velocity based on the collision normal.

These steps allow the grenade to “reflect” from a surface when it collides with one by updating the position and velocity accordingly, using the vector operations provided in the VectorHelper class. Below is an explanation and formula for each of these operations found in the VectorHelper Class.

* **SquareMagnitude(sf::Vector2f vec)**
  + Purpose: Calculates the squared magnitude of a 2D vector.
  + Formula: **vec.x \* vec.x + vec.y \* vec.y**.
  + Explanation: The squared magnitude of a vector is the sum of the squares of its components. This function avoids the square root operation, which can be computationally expensive, by calculating the squared magnitude instead.
* **Magnitude(sf::Vector2f vec)**
  + Purpose: Calculates the magnitude of a 2D vector.
  + Formula: **sqrt(SquareMagnitude(vec))**.
  + Explanation: The magnitude (or length) of a vector is the square root of its squared magnitude. This function uses the **SquareMagnitude** function to calculate the squared magnitude and then takes the square root of it.
* **Normalise(sf::Vector2f vec)**
  + Purpose: Normalizes a 2D vector, making it unit length.
  + Formula: **vec / Magnitude(vec)**.
  + Explanation: To normalize a vector, we divide each component of the vector by its magnitude. This ensures that the vector becomes a unit vector with a length of 1. The function first calculates the magnitude using the **Magnitude** function and then divides each component of the vector by the magnitude.
* **Dot(sf::Vector3f a, sf::Vector3f b)**
  + Purpose: Calculates the dot product of two 3D vectors.
  + Formula: **a.x \* b.x + a.y \* b.y + a.z \* b.z**.
  + Explanation: The dot product of two vectors is the sum of the products of their corresponding components. In 3D, it is calculated by multiplying the corresponding components (**x**, **y**, and **z**) of the two vectors and then summing them up.
* **Dot(sf::Vector2f a, sf::Vector2f b)**
  + Purpose: Calculates the dot product of two 2D vectors.
  + Formula: **a.x \* b.x + a.y \* b.y**.
  + Explanation: The dot product of two vectors is calculated in the same way as in 3D, but in this case, we only consider the **x** and **y** components.
* **Cross(sf::Vector3f a, sf::Vector3f b)**
  + Purpose: Calculates the cross product of two 3D vectors.
  + Formula:
    - **c.x = a.y \* b.z - a.z \* b.y**
    - **c.y = a.z \* b.x - a.x \* b.z**
    - **c.z = a.x \* b.y - a.y \* b.x**
  + Explanation: The cross product of two vectors produces a vector that is perpendicular (orthogonal) to both of the input vectors. In 3D, the cross product is calculated by taking the determinants of matrices formed by the input vectors and the unit vectors (**i**, **j**, **k**) in the **x**, **y**, and **z** directions.
* **GetReflection(sf::Vector3f incident, sf::Vector3f normal)**
  + Purpose: Calculates the reflection of an incident vector off a surface with a given normal in 3D.
  + Formula: **incident - 2.0f \* normal \* Dot(incident, normal)**.
  + Explanation: The reflection of a vector off a surface can be obtained by subtracting twice the projection of the incident vector onto the surface normal from the incident vector itself. This formula uses the dot product to calculate the projection and applies it to the normal vector.
* **GetReflection(sf::Vector2f incident, sf::Vector2f normal)**
  + Purpose: Calculates the reflection of an incident vector off a surface with a given normal in 2D.
  + Formula: **incident - 2.0f \* normal \* Dot(incident, normal)**.
  + Explanation: This function follows the same reflection formula as in 3D, but it operates in 2D space. The dot product is used to calculate the projection of the incident vector onto the normal vector.
* **GetNormal(sf::Vector3f line1, sf::Vector3f line2)**
  + Purpose: Calculates the normal vector of a line defined by two points in 3D.
  + Formula: The normal vector is obtained by taking the cross product of two vectors that lie on the line.
  + Explanation: Given two points that define a line, we can obtain a direction vector by subtracting one point from the other. The normal vector of the line is then calculated by taking the cross product of this direction vector with another vector on the line.
* **GetNormal(sf::Vector2f line)**
  + Purpose: Calculates the normal vector of a line defined by a point in 2D.
  + Formula: The normal vector is obtained by swapping the components and negating one of them.
  + Explanation: In 2D, the normal vector of a line can be obtained by swapping the **x** and **y** components of the direction vector and negating one of them. This is done to ensure that the normal vector is perpendicular to the line.