ゲームプログラミング改

○評価要件

☑ダメージ処理による敵の破棄
☑ダメージ中の無敵時間処理

○概要

今回はダメージ処理を実装します。
今までの課題を全て実装していた場合、敵にダメージを与える手段は２つあります。
敵を踏みつける行為と、弾丸を当てる行為です。

キャラクターに HP を用意し、ダメージを与えて HP が 0 以下になった場合に敵を破棄する処理を実装しましょう。

○健康状態の変更

キャラクターに健康状態を表す変数を用意し、ダメージを与える関数を用意しましょう。

Character.h

```
---省略---

// キャラクター
class Character
{
public:
    ---省略---

    // ダメージを与える
    bool ApplyDamage(int damage);

protected:
    ---省略---

    // ダメージを受けた時に呼ばれる
    virtual void OnDamaged() {}

    // 死亡した時に呼ばれる
    virtual void OnDead() {}

protected:
    ---省略---
    int                     health = 5;
};
```

関数を用意したら実装をしましょう。
ダメージを与える際、様々な状態を判断して派生クラスにダメージ通知や死亡通知関数を呼ぶように実装しましょう。

Character.cpp

```
---省略---

// ダメージを与える
bool Character::ApplyDamage(int damage)
{
    // ダメージが0の場合は健康状態を変更する必要がない
    if (damage == 0) return false;

    // 死亡している場合は健康状態を変更しない
    if (health <= 0) return false;

    // ダメージ処理
    health -= damage;

    // 死亡通知
    if (health <= 0)
    {
```

```
        OnDead();
    }
    // ダメージ通知
    else
    {
        OnDamaged();
    }

    // 健康状態が変更した場合はtrueを返す
    return true;
}

---省略---
```

○敵の破棄処理

健康状態が変更した際、死亡通知が呼び出されるようになったので、派生クラスであるエネミークラスで死亡した時の破棄処理を実装しましょう。
実装方法は基本的に弾丸の時と同じです。

EnemyManager.h

```
---省略---

// エネミーマネージャー
class EnemyManager
{
private:
    ---省略---

public:
    ---省略---

    // エネミー削除
    void Remove(Enemy* enemy);

    ---省略---

private:
    ---省略---
    std::vector<Enemy*>    removes;
};
```

EnemyManager.cpp

```
---省略---

// 更新処理
void EnemyManager::Update(float elapsedTime)
{
    ---省略---

    // 破棄処理
```

```
    // ※enemiesの範囲for文中でerase()すると不具合が発生してしまうため、
    //   更新処理が終わった後に破棄リストに積まれたオブジェクトを削除する。
    for (Enemy* enemy : removes)
    {
        // std::vectorから要素を削除する場合はイテレーターで削除しなければならない
        std::vector<Enemy*>::iterator it = std::find(enemies.begin(), enemies.end(), enemy);
        if (it != enemies.end())
        {
            enemies.erase(it);
        }

        // 削除
        delete enemy;
    }
    // 破棄リストをクリア
    removes.clear();

    // 敵同士の衝突処理
    ---省略---
}

---省略---

// エネミー削除
void EnemyManager::Remove(Enemy* enemy)
{
    // 破棄リストに追加
    removes. emplace_back(enemy);
}

---省略---
```

Enemy.h

```
---省略---

// エネミー
class Enemy : public Character
{
public:
    ---省略---

    // 破棄
    void Destroy();

    ---省略---
};
```

Enemy.cpp

```
---省略---
#include "EnemyManager.h"

---省略---
```

```
// 破棄
void Enemy::Destroy()
{
    EnemyManager::Instance().Remove(this);
}
```

EnemySlime.h

```
---省略---

// スライム
class EnemySlime : public Enemy
{
public:
    ---省略---

protected:
    // 死亡した時に呼ばれる
    void OnDead() override;

    ---省略---
};
```

EnemySlime.cpp

```
---省略---

// 死亡した時に呼ばれる
void EnemySlime::OnDead()
{
    // 自身を破棄
    Destroy();
}
```

○ダメージ処理

次はダメージを与える実装を行いましょう。
現在、プレイヤーに敵を攻撃する手段があります。
弾丸と敵の衝突判定を行い、ダメージ処理を実装しましょう。
現在、弾丸には衝突判定をするための情報が足りないので、情報を追加しましょう。

Projectile.h

```
---省略---

// 弾丸
class Projectile
{
public:
    ---省略---
```

```
    // 半径取得
    float GetRadius() const { return radius; }

    ---省略---

protected:
    ---省略---
    float                   radius = 0.5f;
};
```

○弾丸の衝突形状の可視化

弾丸の衝突処理は球形状で行うことにします。
確認用にデバッグプリミティブも表示できるようにしましょう。

Projectile.cpp

```
---省略---
#include "Graphics/Graphics.h"

---省略---

// デバッグプリミティブ描画
void Projectile::DrawDebugPrimitive()
{
    DebugRenderer* debugRenderer = Graphics::Instance().GetDebugRenderer();

    // 衝突判定用のデバッグ球を描画
    debugRenderer->DrawSphere(position, radius, DirectX::XMFLOAT4(0, 0, 0, 1));
}

---省略---
```

○球と円柱の衝突処理

弾丸と敵の衝突判定を行いますが、前回の課題でキャラクターの衝突判定を円柱に変更した人は球と円柱の衝突判定を実装しておきましょう。

Collision.h

```
---省略---

// コリジョン
class Collision
{
public:
    ---省略---

    // 球と円柱の交差判定
    static bool IntersectSphereVsCylinder(
        const DirectX::XMFLOAT3& spherePosition,
        float sphereRadius,
```

```
        const DirectX::XMFLOAT3& cylinderPosition,
        float cylinderRadius,
        float cylinderHeight,
        DirectX::XMFLOAT3& outCylinderPosition
    );
};
```

Collision.cpp

```
---省略---

// 球と円柱の交差判定
bool Collision::IntersectSphereVsCylinder(
    const DirectX::XMFLOAT3& spherePosition,
    float sphereRadius,
    const DirectX::XMFLOAT3& cylinderPosition,
    float cylinderRadius,
    float cylinderHeight,
    DirectX::XMFLOAT3& outCylinderPosition)
{
    // 高さチェック
    if (spherePosition.y + sphereRadius < cylinderPosition.y) return false;
    if (spherePosition.y - sphereRadius > cylinderPosition.y + cylinderHeight) return false;

    // XZ平面での範囲チェック
    float vx = cylinderPosition.x - spherePosition.x;
    float vz = cylinderPosition.z - spherePosition.z;
    float range = sphereRadius + cylinderRadius;
    float distXZ = sqrtf(vx * vx + vz * vz);
    if (distXZ > range) return false;

    // 球が円柱を押し出す
    vx /= distXZ;
    vz /= distXZ;
    outCylinderPosition.x = spherePosition.x + (vx * range);
    outCylinderPosition.y = cylinderPosition.y;
    outCylinderPosition.z = spherePosition.z + (vz * range);

    return true;
}
```

○弾丸と敵の衝突処理

準備ができたらプレイヤークラス内で弾丸と敵の衝突処理を実装しましょう。

Player.h

```
---省略---

// プレイヤー
class Player : public Character
{
public:
    ---省略---
```

```
    // 弾丸と敵の衝突処理
    void CollisionProjectilesVsEnemies();

    ---省略---
};
```

Player.cpp

```
---省略---

// 更新処理
void Player::Update(float elapsedTime)
{
    ---省略---

    // プレイヤーと敵との衝突処理
    ---省略---

    // 弾丸と敵の衝突処理
    CollisionProjectilesVsEnemies();

    ---省略---
}

---省略---

// 弾丸と敵の衝突処理
void Player::CollisionProjectilesVsEnemies()
{
    EnemyManager& enemyManager = EnemyManager::Instance();

    // 全ての弾丸と全ての敵を総当たりで衝突処理
    int projectileCount = projectileManager.GetProjectileCount();
    int enemyCount = enemyManager.GetEnemyCount();
    for (int i = 0; i < projectileCount; ++i)
    {
        Projectile* projectile = projectileManager.GetProjectile(i);

        for (int j = 0; j < enemyCount; ++j)
        {
            Enemy* enemy = enemyManager.GetEnemy(j);

            // 衝突処理
            DirectX::XMFLOAT3 outPosition;
            if (Collision::IntersectSphereVsCylinder(
                projectile->GetPosition(),
                projectile->GetRadius(),
                enemy->GetPosition(),
                enemy->GetRadius(),
                enemy->GetHeight(),
                outPosition))
            {
                // ダメージを与える
                enemy->ApplyDamage(1);
```

このタイミングで弾丸も
消滅させても良いが、
次の学習のため今はあえてしない

```
            }
        }
    }
}
```

踏みつけ処理も実装している場合はそちらでもダメージ処理を実装しましょう。

ここまで実装したら実行確認をしてみましょう。
弾丸を発射し、敵に何発か当てて敵が消滅すると OK です。

○無敵時間の設定

現在、敵に弾を当てると敵の消滅処理が実装出来ていますが、正しいダメージ処理ができていません。
課題通りの実装だと、敵の健康状態の値は５なので５回弾丸を当てる必要があるのですが、１～２回で消滅していると思います。
これは衝突範囲にいるときは毎フレームダメージを与えているからです。
こういう場合にゲームプログラミングでは無敵時間を設けて無敵時間以内に衝突してもダメージを受けないという実装をします。
ダメージ処理を改造して無敵時間対応を行いましょう。

Charcter.h

```
---省略---

// キャラクター
class Character
{
public:
    ---省略---

    // ダメージを与える
    bool ApplyDamage(int damage);
    bool ApplyDamage(int damage, float invincibleTime);

    ---省略---

protected:

    ---省略---

    // 無敵時間更新
    void UpdateInvincibleTimer(float elapsedTime);

    ---省略---

protected:
    ---省略---
    float                   invincibleTimer = 1.0f;
```

```
};
```

Character.cpp

```
---省略---

// ダメージを与える
bool Character::ApplyDamage(int damage)
bool Character::ApplyDamage(int damage, float invincibleTime)
{
    ---省略---

    // 死亡している場合は健康状態を変更しない
    ---省略---

    // 無敵時間中はダメージを与えない
    if (invincibleTimer > 0.0f) return false;

    // 無敵時間設定
    invincibleTimer = invincibleTime;

    // ダメージ処理
    ---省略---

    // 健康状態が変更した場合はtrueを返す
    return true;
}

---省略---

// 無敵時間更新
void Character::UpdateInvincibleTimer(float elapsedTime)
{
    if (invincibleTime > 0.0f)
    {
        invincibleTime -= elapsedTime;
    }
}
```

ダメージを与えたときに
無敵時間を設定し、
無敵中はダメージを
与えないようにしよう

EnemySlime.cpp

```
---省略---

// 更新処理
void EnemySlime::Update(float elapsedTime)
{
    ---省略---

    // 無敵時間更新
    UpdateInvincibleTimer(elapsedTime);

    // オブジェクト行列を更新
    ---省略---
}
```

Player.cpp

```
---省略---

// 弾丸と敵の衝突処理
void Player::CollisionProjectilesVsEnemies()
{
    ---省略---
    for (int i = 0; i < projectileCount; ++i)
    {
        ---省略---
        for (int j = 0; j < enemyCount; ++j)
        {
            ---省略---
            // 衝突処理
            ---省略---
            if (Collision::IntersectSphereVsCylinder(---省略---))
            {
                // ダメージを与える
                enemy->ApplyDamage(1);
                enemy->ApplyDamage(1, 0.5f);
            }
        }
    }
}
```

実装出来たら実行確認してみて下さい。
弾丸を５発当てて敵が消滅できれば OK です。

○衝突後の弾丸の破棄

無敵時間の学習のために衝突した時に弾丸を破棄していませんでしたが、弾丸も破棄するようにしておきましょう。

Player.cpp

```
---省略---

// 弾丸と敵の衝突処理
void Player::CollisionProjectilesVsEnemies()
{
    ---省略---
    for (int i = 0; i < projectileCount; ++i)
    {
        ---省略---
        for (int j = 0; j < enemyCount; ++j)
        {
            ---省略---
            // 衝突処理
            ---省略---
```

```
            if (Collision::IntersectSphereVsCylinder(---省略---))
            {
                // ダメージを与える
                enemy->ApplyDamage(1, 0.5f);
                if (enemy->ApplyDamage(1, 0.5f))
                {
                    // 弾丸破棄
                    projectile->Destroy();
                }
            }
        }
    }
}
```

お疲れさまでした。