ゲームプログラミング改

○評価要件

☑Effekseer の導入
☑エフェクトの再生＆表示処理
☑エディタを使用して自作のエフェクトの表示

○概要

今回はエフェクトを実装します。
エフェクトはゲームでは欠かせない要素の一つです。
ビジュアルの強化はもちろん、ゲーム内で起きた現象をユーザーにわかりやすく伝えるという目的もあります。

エフェクトシステムを実装することは大変です。
システムの作成はもちろんですが、エフェクトを作成するエディタも必要になってきます。
これを自作で作ることができたら自身の大きなアピールポイントになるでしょう。

今回は手軽にエフェクトシステムを組み込むために「Effekseer」（エフェクシア）というライブラリを導入してゲームでのエフェクトの実装方法を体験していきます。

グラフィックスに興味のある人は Effekseer を参考にして将来的に自作のエフェクトエンジンを作成しましょう。

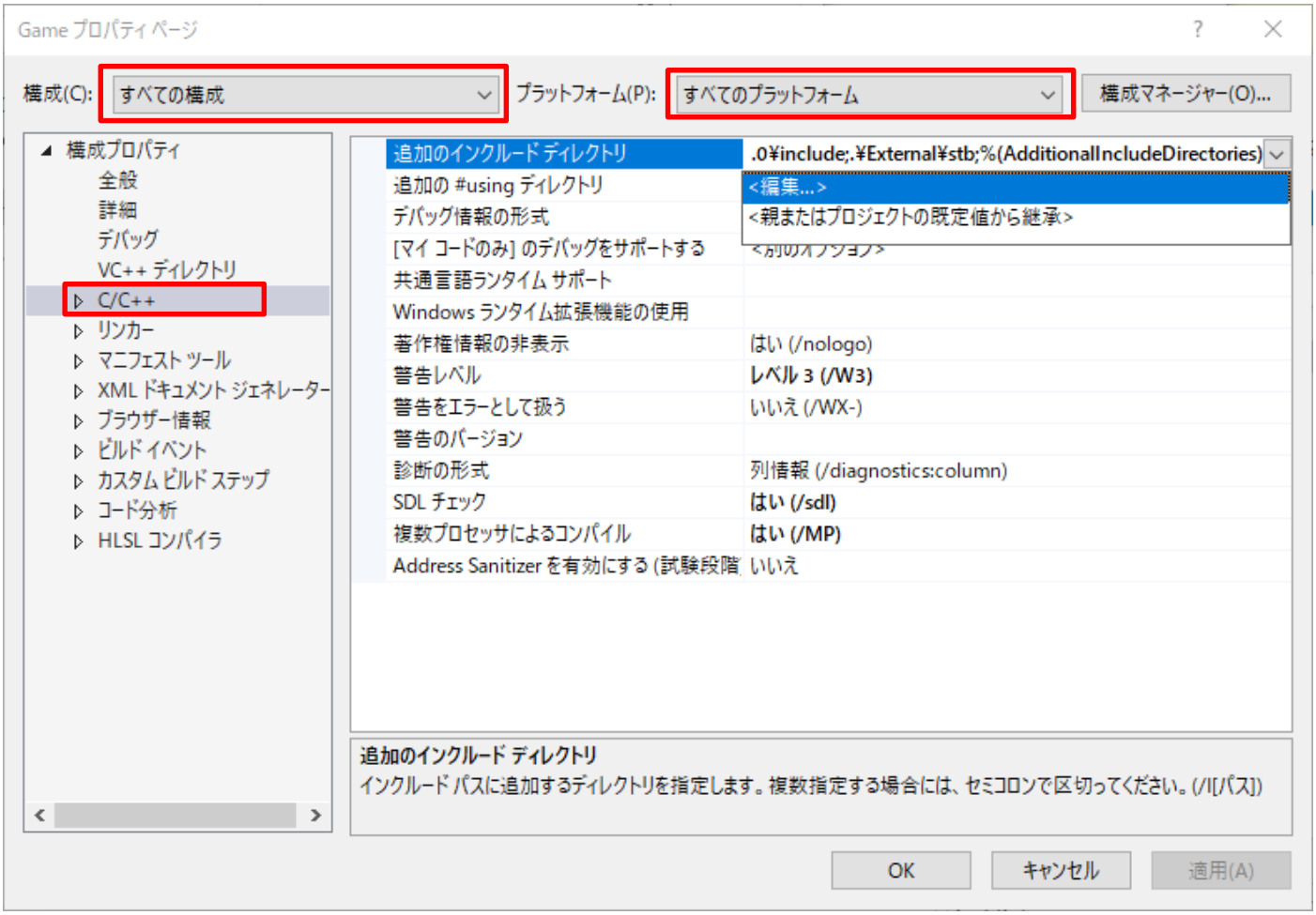
○Effekseer 導入

Effekseer とはオープンソースのエフェクトライブラリです。
制作者が日本人で市販のゲームにも採用されている強力なエフェクトシステムです。

Effekseer の公式ホームページからエディタとライブラリをダウンロードできます。
https://effekseer.github.io/jp/

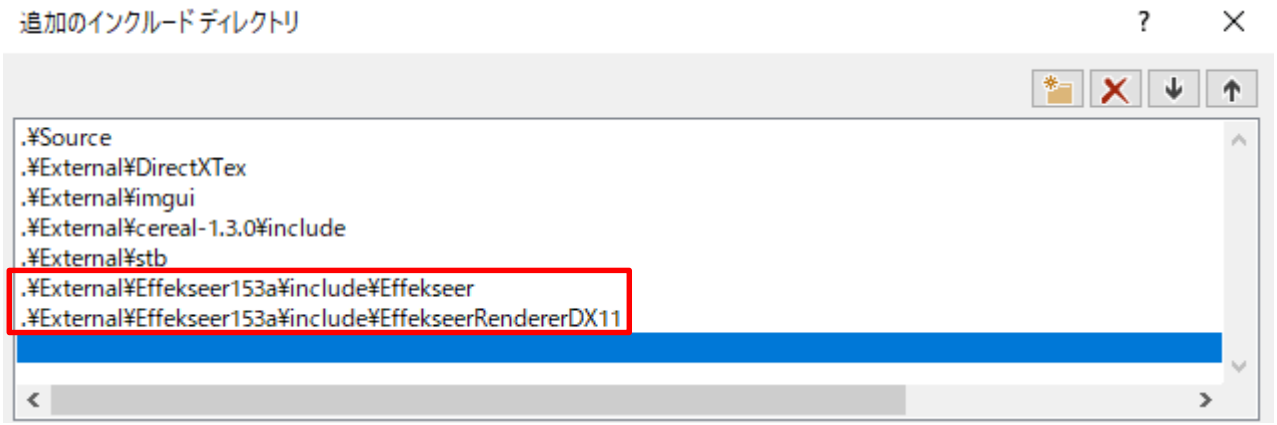
本来はライブラリをダウンロードし、readme.txt に記述されている通りにライブラリをコンパイルし、ライブラリファイル（.lib）を作成して導入しますが、コンパイルするまでに必要なツールをインストールするなど、各自の環境でトラブルが起きることが予想されるので課題ではコンパイル済みのライブラリを提供します。
自信のある人は今後、外部ライブラリを利用することも多々あるので練習としてコンパイルも自分でやってみましょう。

VisualStudio のプロジェクト設定で左上の「構成」を「すべての構成」、「プラットフォーム」を「すべてのプラットフォーム」に設定し、「C/C++」を選択し、「追加のインクルードディレクトリ」を編集しましょう。



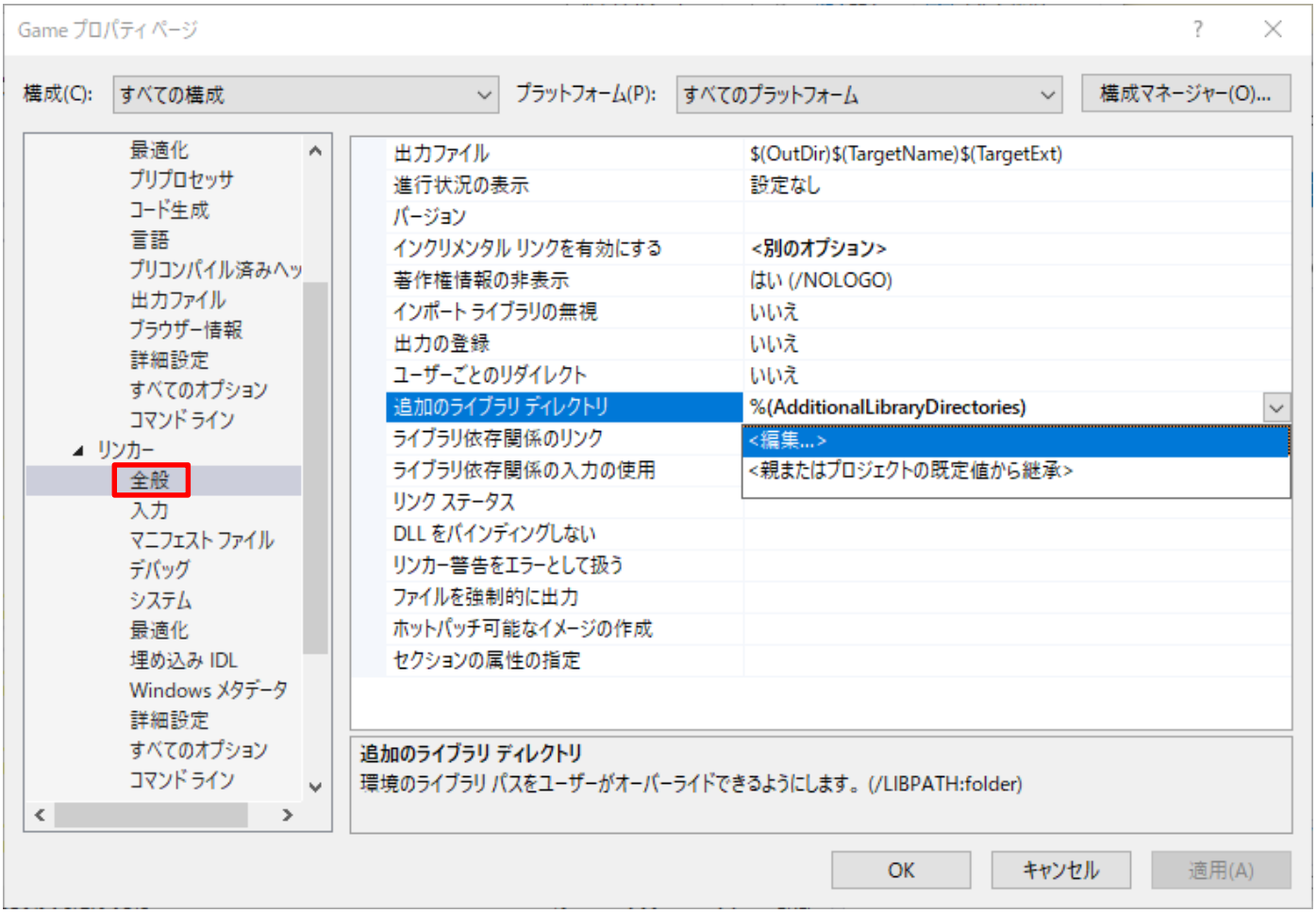
追加のインクルードディレクトリ編集画面で以下の２つのパスを追加しましょう。

```
.¥External¥Effekseer153a¥include¥Effekseer
.¥External¥Effekseer153a¥include¥EffekseerRendererDX11
```



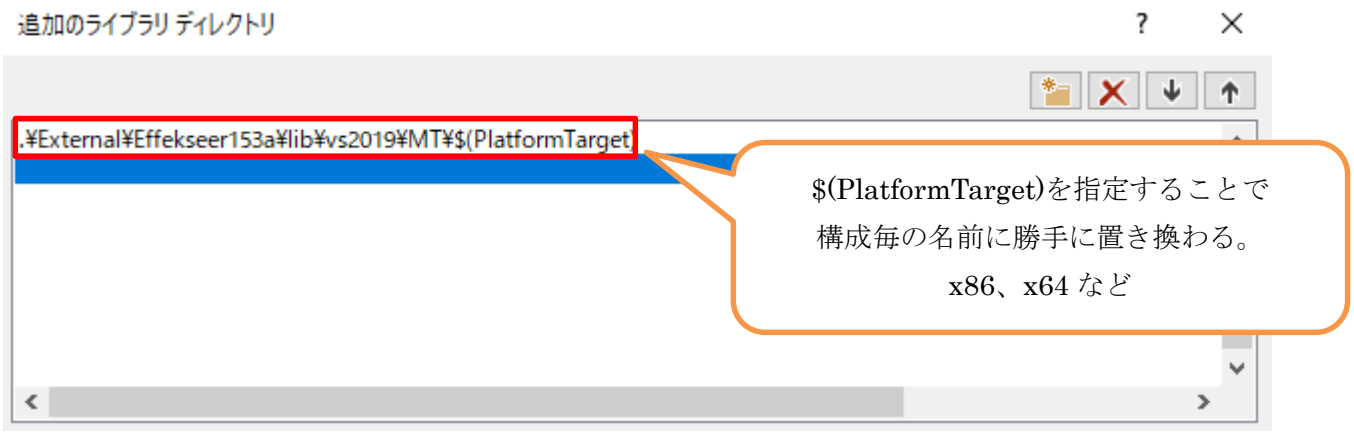
これで Effekseer をプログラムで実装する準備ができました。

次はライブラリのリンク設定をしましょう。
プロジェクト設定画面で「リンカー」→「全般」を選択し、「追加のライブラリディレクトリ」を編集しましょう。

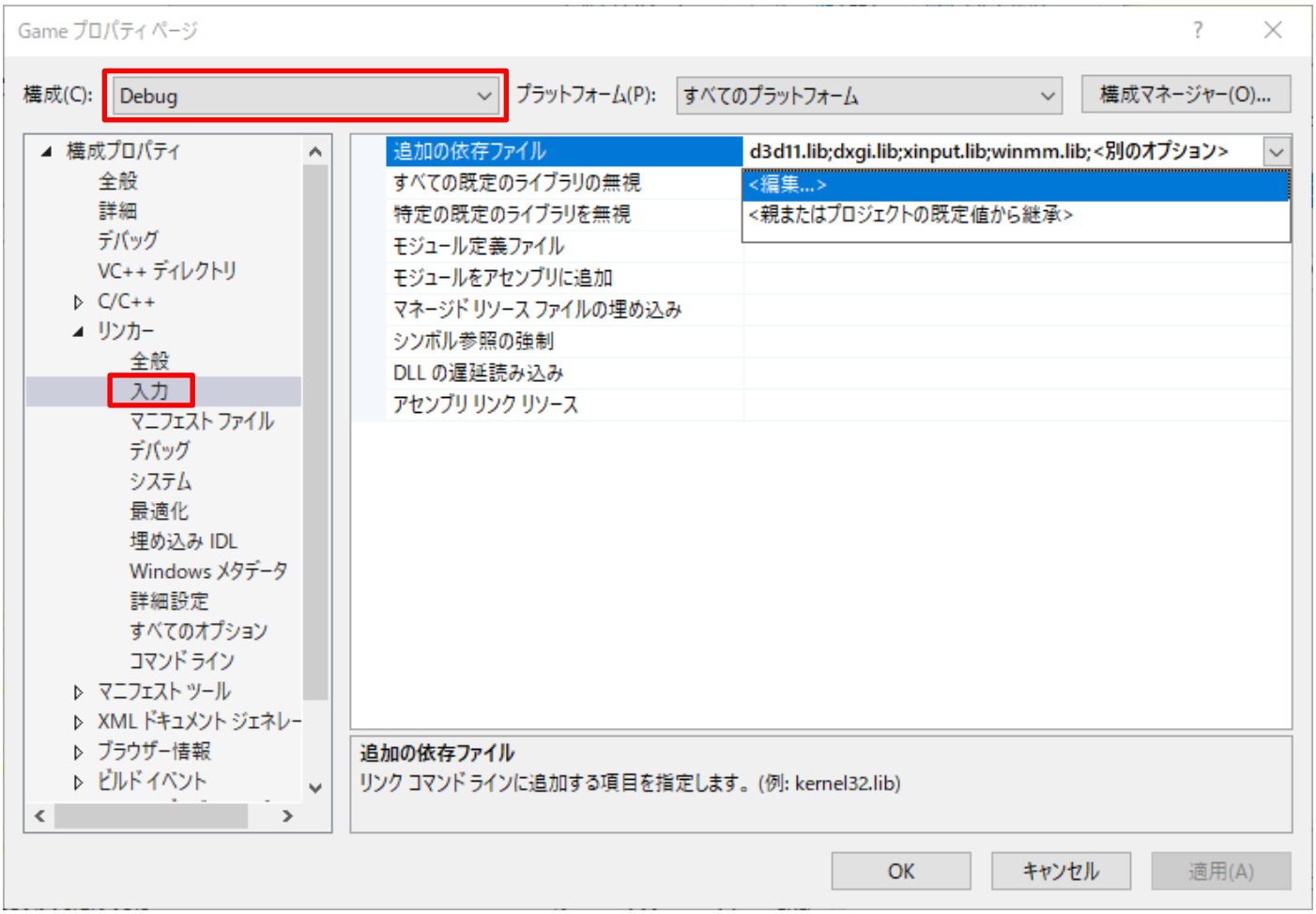


追加のライブラリディレクトリ編集画面で以下のパスを追加しましょう。

.¥External¥Effekseer153a¥lib¥vs2019¥MT¥$(PlatformTarget)

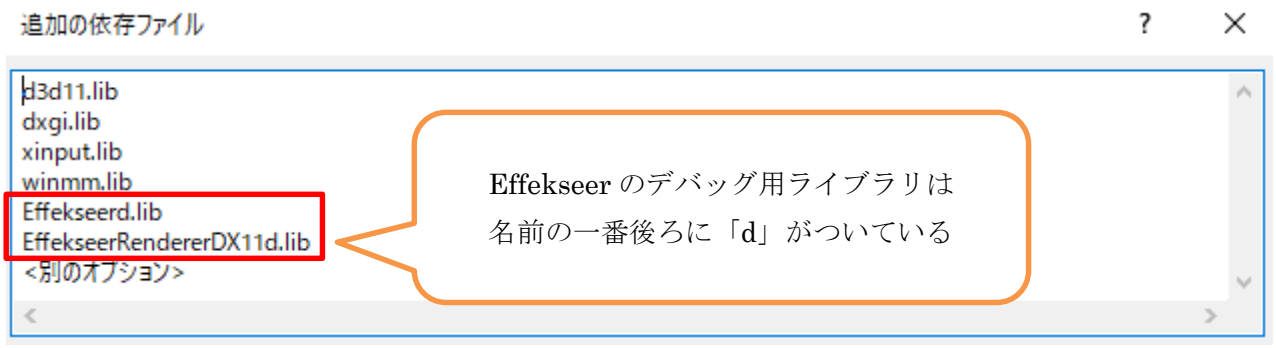


次はリンクするライブラリを指定しましょう
「構成」を「Debug」に設定し、「リンカ」→「入力」を選択し、「追加の依存ファイル」を編集しましょう。

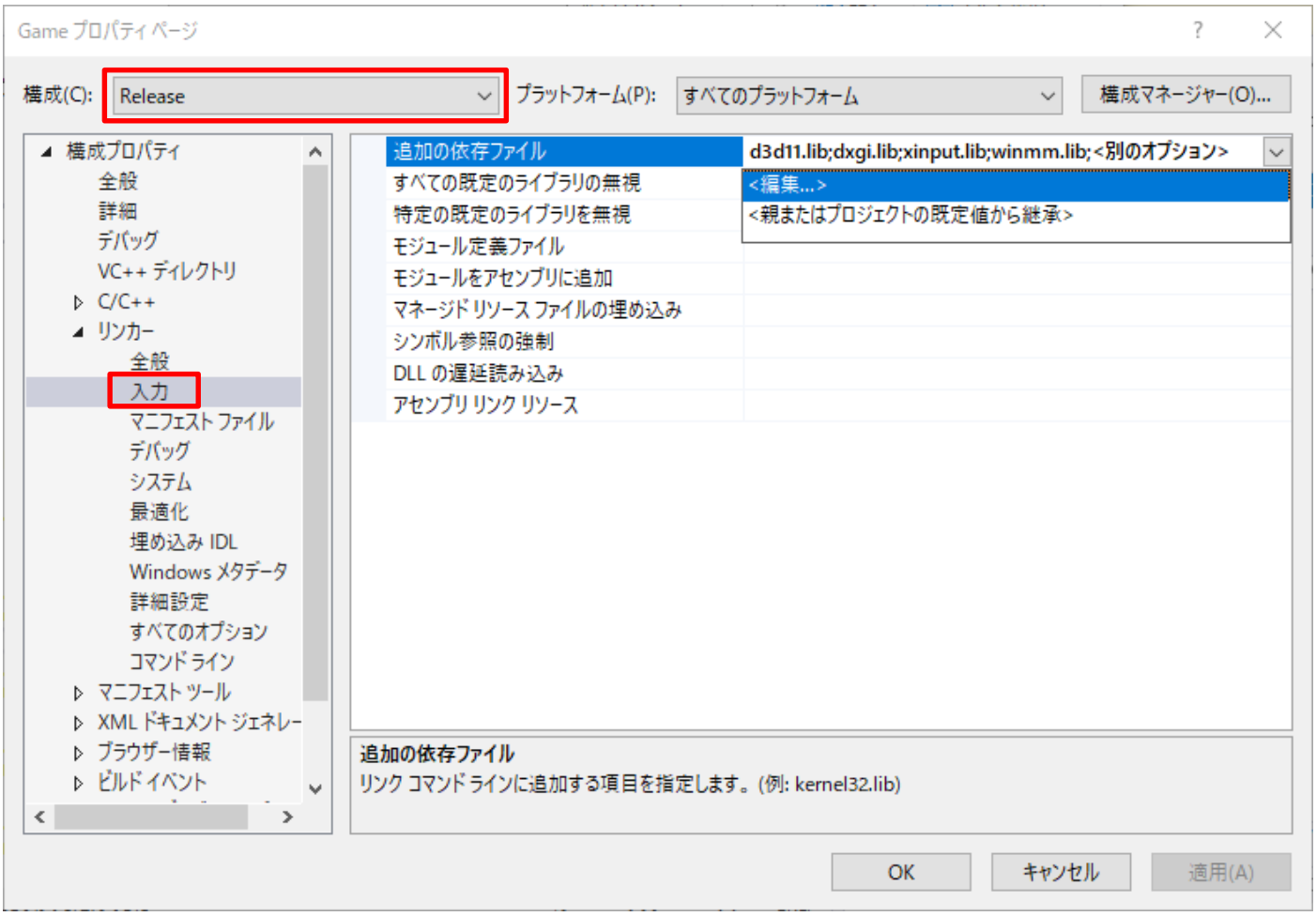


追加の依存ファイル編集画面で以下のライブラリを追加しましょう

Effekseerd.lib EffekseerRendererDX11d.lib

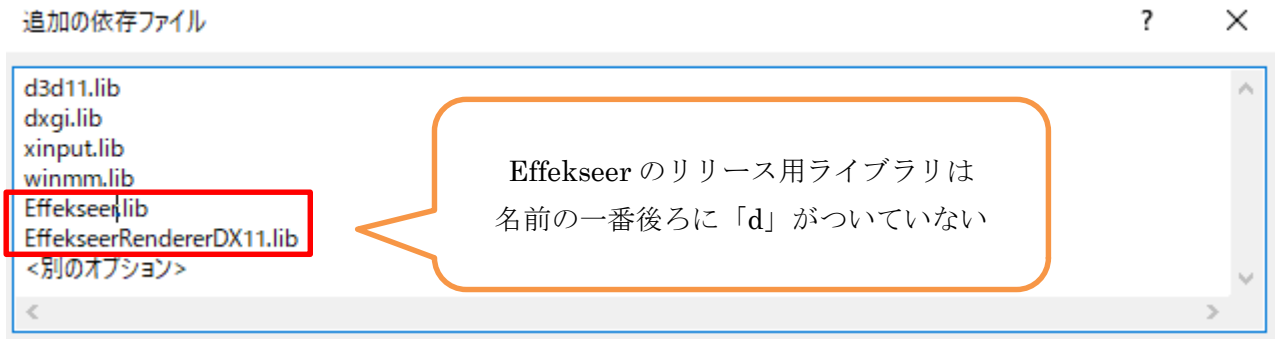


続いて「Release」も設定しましょう。
「構成」を「Release」に設定し、「リンカー」→「入力」を選択し、「追加の依存ファイル」を編集しましょう。



追加の依存ファイル編集画面で以下の２つのライブラリを追加しましょう。

Effekseer.lib EffekseerRendererDX11.lib



これで Effekseer の導入は完了しました。

○Effekseer の解説

Effekseer は大きく分けて４つの重要なクラスが存在します。

クラス	説明
EffekseerRenderer::Renderer	エフェクトを描画するクラス
Effekseer::Manager	エフェクトリソースとインスタンスを管理するクラス
Effeksser::Effect	エフェクトリソースからインスタンスを生成するクラス
Effekseer::Handle	エフェクトインスタンスを操作するためのハンドル

EffekseerRenderer::Renderer はエフェクトを描画するクラスです。
このクラスはゲーム内で１つだけあれば十分です。
このクラスに描画に必要なカメラ情報などを渡し、BeginRendering()と EndRendering()関数の間で描画処理を実行することでエフェクトを描画できます。

Effekseer::Manager はエフェクトリソースとインスタンスを管理するクラスです。
このマネージャーに登録されているエフェクトを一括で描画実行したり、エフェクトインスタンス毎の操作を行ったりします。
このクラスは描画順の制御やシーン毎に別で管理したい場合は複数作成する場合がありますが、今回は一つだけしか作成しません。

Effekseer::Effect はエフェクトを発生させるクラスです。
ファイルパスから読み込んだエフェクトデータを Play()関数を呼ぶことでエフェクトインスタンスを生成します。このクラスはエフェクトファイル毎に作成されます。

Effekseer::Handle はエフェクトのインスタンスハンドルです。
Effekseer::Manager にこのハンドルを渡すことでエフェクトの再生速度を設定やエフェクトを停止するなどの操作を行えます。
このハンドルはゲーム上に発生するエフェクトの数だけ作成されます。

○Effekseer の実装

まずは Effekseer の初期設定をしましょう。
今回はエフェクト全体を管理する EffectManager クラスを作成します。
EffectManager.cpp と EffectManager.h を作成し、下記プログラムコードを記述しましょう。

EffectManager.h

```
#pragma once

#include <DirectXMath.h>
#include <Effekseer.h>
#include <EffekseerRendererDX11.h>

// エフェクトマネージャー
class EffectManager
{
private:
    EffectManager() {}
    ~EffectManager() {}

public:
    // 唯一のインスタンス取得
    static EffectManager& Instance()
    {
        static EffectManager instance;
        return instance;
    }

    // 初期化
    void Initialize();

    // 終了化
    void Finalize();

    // 更新処理
    void Update(float elapsedTime);

    // 描画処理
    void Render(const DirectX::XMFLOAT4X4& view, const DirectX::XMFLOAT4X4& projection);

    // Effeckseerマネージャーの取得
    Effekseer::Manager* GetEffekseerManager() { return effekseerManager; }

private:
    Effekseer::Manager*           effekseerManager = nullptr;
    EffekseerRenderer::Renderer*  effekseerRenderer = nullptr;
```

```
};
```

EffectManager.cpp

```
#include "Graphics/Graphics.h"
#include "EffectManager.h"

// 初期化
void EffectManager::Initialize()
{
    Graphics& graphics = Graphics::Instance();

    // Effekseerレンダラ生成
    effekseerRenderer = EffekseerRendererDX11::Renderer::Create(graphics.GetDevice(),
                                                                graphics.GetDeviceContext(), 2048);

    // Effekseerマネージャー生成
    effekseerManager = Effekseer::Manager::Create(2048);

    // Effekseerレンダラの各種設定（特別なカスタマイズをしない場合は定型的に以下の設定でOK）
    effekseerManager->SetSpriteRenderer(effekseerRenderer->CreateSpriteRenderer());
    effekseerManager->SetRibbonRenderer(effekseerRenderer->CreateRibbonRenderer());
    effekseerManager->SetRingRenderer(effekseerRenderer->CreateRingRenderer());
    effekseerManager->SetTrackRenderer(effekseerRenderer->CreateTrackRenderer());
    effekseerManager->SetModelRenderer(effekseerRenderer->CreateModelRenderer());
    // Effekseer内でのローダーの設定（特別なカスタマイズをしない場合は以下の設定でOK）
    effekseerManager->SetTextureLoader(effekseerRenderer->CreateTextureLoader());
    effekseerManager->SetModelLoader(effekseerRenderer->CreateModelLoader());
    effekseerManager->SetMaterialLoader(effekseerRenderer->CreateMaterialLoader());

    // Effekseerを左手座標系で計算する
    effekseerManager->SetCoordinateSystem(Effekseer::CoordinateSystem::LH);
}

// 終了化
void EffectManager::Finalize()
{
    if (effekseerManager != nullptr)
    {
        effekseerManager->Destroy();
        effekseerManager = nullptr;
    }
    if (effekseerRenderer != nullptr)
    {
        effekseerRenderer->Destroy();
        effekseerRenderer = nullptr;
    }
}

// 更新処理
void EffectManager::Update(float elapsedTime)
{
    // エフェクト更新処理（引数にはフレームの経過時間を渡す）
    effekseerManager->Update(elapsedTime * 60.0f);
}
```

```
// 描画処理
void EffectManager::Render(const DirectX::XMFLOAT4X4& view, const DirectX::XMFLOAT4X4&
projection)
{
    // ビュー＆プロジェクション行列をEffekseerレンダラに設定
    effekseerRenderer->SetCameraMatrix(*reinterpret_cast<const Effekseer::Matrix44*>(&view));
    effekseerRenderer->SetProjectionMatrix(*reinterpret_cast<const
Effekseer::Matrix44*>(&projection));

    // Effekseer描画開始
    effekseerRenderer->BeginRendering();

    // Effekseer描画実行
    // マネージャー単位で描画するので描画順を制御する場合はマネージャーを複数個作成し、
    // Draw()関数を実行する順序で制御できそう
    effekseerManager->Draw();

    // Effekseer描画終了
    effekseerRenderer->EndRendering();
}
```

基本的な Effekseer の描画についての実装は以上で完了です。
EffectManager はゲーム中に一度だけ初期化するだけで良いので Framework のクラスのコンストラクタで呼び出しましょう。

Framework.cpp

```
---省略---

#include "EffectManager.h"

---省略---

// コンストラクタ
Framework::Framework(HWND hWnd)
    : hWnd(hWnd)
    , graphics(hWnd)
{
    // エフェクトマネージャー初期化
    EffectManager::Instance().Initialize();

    // シーン初期化
    sceneGame.Initialize();
}

// デストラクタ
Framework::~Framework()
{
    // シーン終了化
    sceneGame.Finalize();

    // エフェクトマネージャー終了化
    EffectManager::Instance().Finalize();
```

```
}
---省略---
```

エフェクトマネージャーの初期化と終了化を実装しました。
描画処理の方はシーン毎に呼び出す順番を制御する場合もシーンの更新処理、描画処理で呼び出すようにしましょう。

SceneGame.cpp

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

---省略---

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

    // エフェクト更新処理
    EffectManager::Instance().Update(elapsedTime);
}

// 描画処理
void SceneGame::Render()
{
    ---省略---

    // 3Dモデル描画
    {
        ---省略---
    }

    // 3Dエフェクト描画
    {
        EffectManager::Instance().Render(rc.view, rc.projection);
    }

    ---省略---
}
```

これでエフェクトの描画システムは実装できました。
後はエフェクトを読み込み、再生すれば表示できます。
Effekseer のエフェクトを扱いやすくするエフェクトクラスを作成しましょう。
Effect.cpp と Effect.h を作成し、下記プログラムコードを記述してください。

Effect.h

```
#pragma once

#include <DirectXMath.h>
#include <Effekseer.h>
```

```
// エフェクト
class Effect
{
public:
    Effect(const char* filename);
    ~Effect();

    // 再生
    Effekseer::Handle Play(const DirectX::XMFLOAT3& position, float scale = 1.0f);

    // 停止
    void Stop(Effekseer::Handle handle);

    // 座標設定
    void SetPosition(Effekseer::Handle handle, const DirectX::XMFLOAT3& position);

    // スケール設定
    void SetScale(Effekseer::Handle handle, const DirectX::XMFLOAT3& scale);

private:
    Effekseer::Effect* effekseerEffect = nullptr;
};
```

Effect.cpp

```
#include "Graphics/Graphics.h"
#include "Effect.h"
#include "EffectManager.h"

// コンストラクタ
Effect::Effect(const char* filename)
{
    // Effekseerのリソースを読み込む
    // EffekseerはUTF-16のファイルパス以外は対応していないため文字コード変換が必要
    char16_t utf16Filename[256];
    Effekseer::ConvertUtf8ToUtf16(utf16Filename, 256, filename);

    // Effekseer::Managerを取得
    Effekseer::Manager* effekseerManager = EffectManager::Instance().GetEffekseerManager();

    // Effekseerエフェクトを読み込み
    effekseerEffect = Effekseer::Effect::Create(effekseerManager, (EFK_CHAR*)utf16Filename);
}

// デストラクタ
Effect::~Effect()
{
    // 破棄処理
    if (effekseerEffect != nullptr)
    {
        effekseerEffect->Release();
        effekseerEffect = nullptr;
    }
}
```

```
// 再生
Effekseer::Handle Effect::Play(const DirectX::XMFLOAT3& position, float scale)
{
    Effekseer::Manager* effekseerManager = EffectManager::Instance().GetEffekseerManager();

    Effekseer::Handle handle = effekseerManager->Play(effekseerEffect, position.x, position.y,
position.z);
    effekseerManager->SetScale(handle, scale, scale, scale);
    return handle;
}

// 停止
void Effect::Stop(Effekseer::Handle handle)
{
    Effekseer::Manager* effekseerManager = EffectManager::Instance().GetEffekseerManager();

    effekseerManager->StopEffect(handle);
}

// 座標設定
void Effect::SetPosition(Effekseer::Handle handle, const DirectX::XMFLOAT3& position)
{
    Effekseer::Manager* effekseerManager = EffectManager::Instance().GetEffekseerManager();

    effekseerManager->SetLocation(handle, position.x, position.y, position.z);
}

// スケール設定
void Effect::SetScale(Effekseer::Handle handle, const DirectX::XMFLOAT3& scale)
{
    Effekseer::Manager* effekseerManager = EffectManager::Instance().GetEffekseerManager();

    effekseerManager->SetScale(handle, scale.x, scale.y, scale.z);
}
```

エフェクトクラスはエフェクトデータを読み込み、再生を行えるようにしました。
再生して生成されたエフェクトインスタンスハンドルを保持しておくと、座標やスケールなどの各種設定を個別に設定できます。

今回は座標とスケールの設定をできるようにしましたが、他にも色々できるので自分で Effekseer の関数を調べて扱いやすいエフェクトクラスに拡張していきましょう。

次はいよいよエフェクトの再生です。
前回までの課題で弾丸と敵の衝突判定ができているので、敵にダメージを与えたタイミングでヒットエフェクトが発生するように実装してみましょう。

Player.h

```
---省略---

#include "Effect.h"

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

private:
    ---省略---
    Effect*             hitEffect = nullptr;
};
```

Player.cpp

```
---省略---

// コンストラクタ
Player::Player()
{
    ---省略---

    // ヒットエフェクト読み込み
    hitEffect = new Effect("Data/Effect/Hit.efk");
}

// デストラクタ
Player::~Player()
{
    delete hitEffect;

    ---省略---
}

// 弾丸と敵の衝突処理
void Player::CollisionProjectilesVsEnemies()
{
    ---省略---
    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(---省略---))
            {
                // ダメージを与える
                if (enemy->ApplyDamage(1, 0.5f))
                {
                    // 吹き飛ばす
```

```
                    {
                        ---省略---
                    }

                    // ヒットエフェクト再生
                    {
                        DirectX::XMFLOAT3 e = enemy->GetPosition();
                        e.y += enemy->GetHeight() * 0.5f;
                        hitEffect->Play(e);
                    }

                    // 弾丸破棄
                    ---省略---
                }
            }
        }
    }
}
```

実装出来たら実行確認してみましょう。
弾丸を敵に当ててエフェクトが再生できていれば OK です。

○自作エフェクトの作成

今回は課題で用意しているエフェクトデータを使用しましたが、Effekseer ではエフェクトを作成するためのエディタが用意されています。
基本的にエフェクトデータはエディタを使ってアーティストが用意してくれますが、プログラマはエフェクトデータがどのようにできているか、どうやってエフェクトを作成するのかなどを知っておく必要があります。

Effekseer のエディタを使用して自作のエフェクトを作成し、ゲームに組み込みましょう。

自作エフェクトをゲームに組み込み、表示できたら課題は完了です。
お疲れさまでした。