ECS_BASE

構築: Doxygen 1.14.0

Chapter 1

名前空間索引

1.1 名前空間一覧

全名前空間の一覧です。

Chapter 2

階層索引

2.1 クラス階層

クラス階層一覧です。大雑把に文字符号順で並べられています。

Chapter 3

クラス索引

3.1 クラス一覧

クラス・構造体・共用体・インターフェースの一覧です。

Chapter 4

ファイル索引

4.1 ファイル一覧

ファイル一覧です。

Chapter 5

名前空間詳解

5.1 SampleScenes 名前空間

関数

- Entity CreateSimpleCube (World &world)
  レベル 1: 最もシンプルなエンティティ
- Entity CreateRotatingCube (World &world, const DirectX::XMFLOAT3 &position)
  レベル 2: 動きのあるエンティティ
- Entity CreateBouncingCube (World &world)
  レベル 3: カスタムBehaviour を使う
- Entity CreateComplexCube (World &world)
  レベル 4: 複数のBehaviour を組み合わせる
- Entity CreateCubeOldStyle (World &world)
  レベル 5: 従来の方法でエンティティを作成
- void ModifyEntityExample (World &world, Entity entity)
  レベル 6: コンポーネントの後からの変更
- void ProcessAllTransforms (World &world)
  レベル 7: 全エンティティに対する処理
- void ChangeAllColors (World &world)
  全MeshRenderer の色を変更
- void CreateGridOfCubes (World &world, int rows=3, int cols=3)
  レベル 8: デモシーン作成
- Entity CreateRainbowCube (World &world)
  練習 1: 虹色に回転するキューブを作成
- Entity CreateWanderingCube (World &world)
  練習 2: ランダムに動き回るキューブ
- Entity CreateTemporaryCube (World &world, float lifeTime=5.0f)
  練習 3: 時間経過で消えるキューブ

5.1.1 関数詳解

5.1.1.1 ChangeAllColors()

void SampleScenes::ChangeAllColors (
World & world) [inline]

全MeshRenderer の色を変更

引数

in,out	world	ワ

著者

山内陽

呼び出し関係図:

5.1.1.2 CreateBouncingCube()

Entity SampleScenes::CreateBouncingCube (
World & world) [inline]

レベル 3: カスタムBehaviour を使う

学べること:

- ComponentSamples.h のカスタムBehaviour を使う
- 複数のコンポーネントを組み合わせる

上下に跳ねる黄色のキューブを作成

引数

in,out	world	ワ

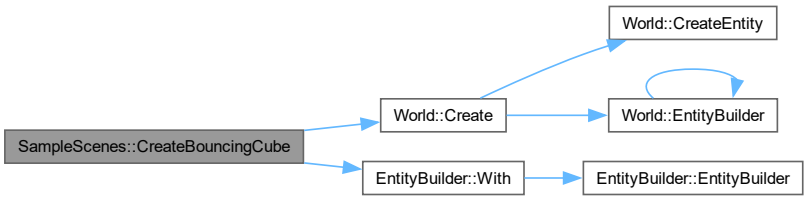
戻り値

Entity 作成されたエンティティ

著者

山内陽

呼び出し関係図:



5.1.1.3 CreateComplexCube()

Entity SampleScenes::CreateComplexCube (
World & world) [inline]

レベル 4: 複数のBehaviour を組み合わせる

学べること:

- 1 つのエンティティに複数のBehaviour を追加
- それぞれが独立して動作する

回転しながらサイズが変わるマゼンタのキューブを作成

引数

	in,out	world	ワ

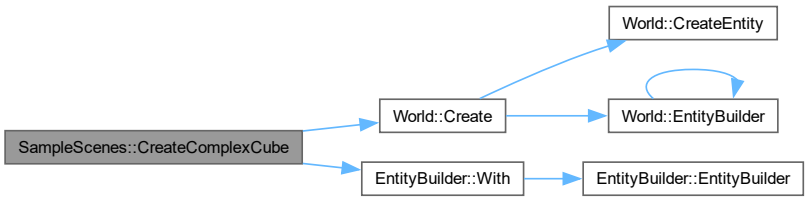
戻り値

Entity 作成されたエンティティ

著者

山内陽

呼び出し関係図:



5.1.1.4 CreateCubeOldStyle()

```
Entity SampleScenes::CreateCubeOldStyle (
            World & world)   [inline]
```

レベル 5: 従来の方法でエンティティを作成

学べること:

- ビルダーパターンを使わない方法
- 手動でコンポーネントを追加する方法

従来の方法でシアンのキューブを作成

引数

in,out	world	ワ

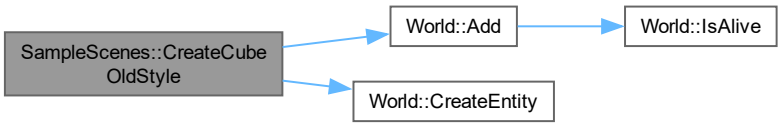
戻り値

Entity 作成されたエンティティ

著者

山内陽

呼び出し関係図:



5.1.1.5 CreateGridOfCubes()

```
void SampleScenes::CreateGridOfCubes (
            World & world,
            int rows = 3,
            int cols = 3)   [inline]
```

レベル 8: デモシーン作成

学べること:

- 複数のエンティティを配置してシーンを構成
- 位置を計算してグリッド状に配置

グリッド状にキューブを配置

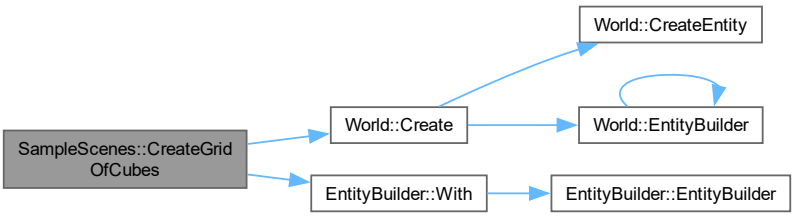
引数

in,out	world	ワ
in	rows	
in	cols	

著者

山内陽

呼び出し関係図:



5.1.1.6 CreateRainbowCube()

Entity SampleScenes::CreateRainbowCube (
World & world) [inline]

練習 1: 虹色に回転するキューブを作成

引数

	in,out	world	ワ

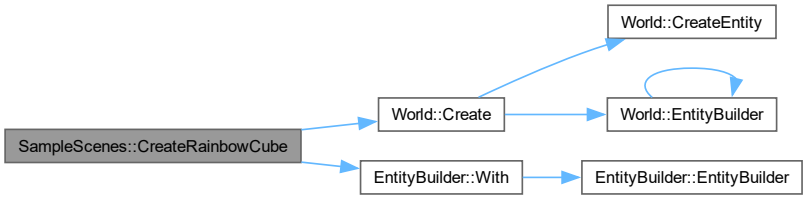
戻り値

Entity 作成されたエンティティ

著者

山内陽

呼び出し関係図:



5.1.1.7 CreateRotatingCube()

Entity SampleScenes::CreateRotatingCube (
World & world,
const DirectX::XMFLOAT3 & position) [inline]

レベル 2: 動きのあるエンティティ

学べること:

- Behaviour コンポーネント（Rotator）の使い方
- コンポーネントの組み合わせ

回転する緑のキューブを作成

引数

in,out	world	
in	position	

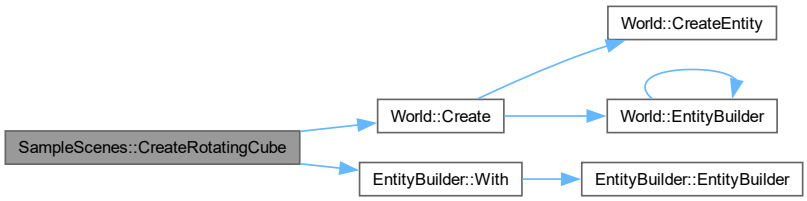
戻り値

Entity 作成されたエンティティ

著者

山内陽

呼び出し関係図:



5.1.1.8 CreateSimpleCube()

Entity SampleScenes::CreateSimpleCube (
World & world) [inline]

レベル 1: 最もシンプルなエンティティ

学べること:

- エンティティの作成方法
- Transform コンポーネントの設定
- MeshRenderer で色を付ける

シンプルな赤いキューブを作成

引数

in,out	world	ワ

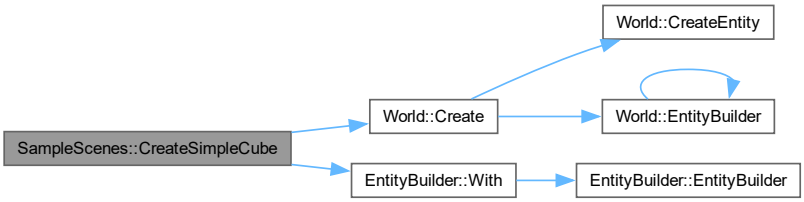
戻り値

Entity 作成されたエンティティ

著者

山内陽

呼び出し関係図:



5.1.1.9 CreateTemporaryCube()

Entity SampleScenes::CreateTemporaryCube (
World & world,
float lifeTime = 5.0f) [inline]

練習 3: 時間経過で消えるキューブ

引数

in,out	world
in	lifeTime

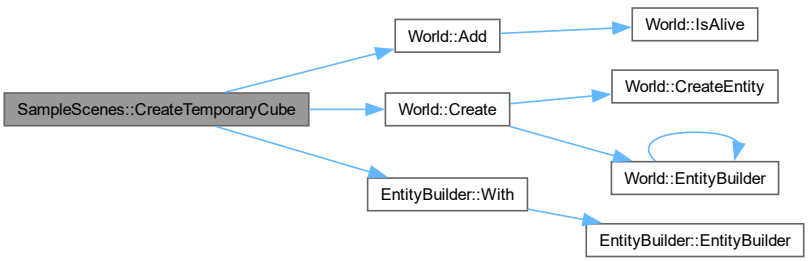
戻り値

Entity 作成されたエンティティ

著者

山内陽

呼び出し関係図:



5.1.1.10 CreateWanderingCube()

Entity SampleScenes::CreateWanderingCube (
World & world) [inline]

練習 2: ランダムに動き回るキューブ

引数

in,out	world	ワ

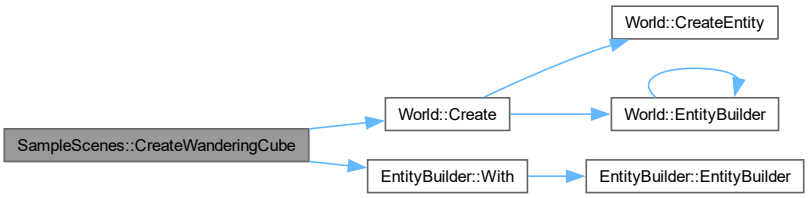
戻り値

Entity 作成されたエンティティ

著者

山内陽

呼び出し関係図:



5.1.1.11 ModifyEntityExample()

void SampleScenes::ModifyEntityExample (
World & world,
Entity entity) [inline]

レベル 6: コンポーネントの後からの変更

学べること:

- TryGet で取得して値を変更
- コンポーネントの動的な操作

エンティティのコンポーネントを変更する例

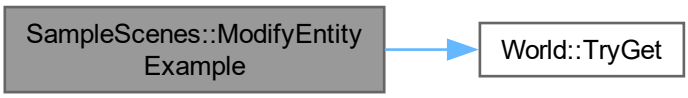
引数

in,out	world	
in	entity	変

著者

山内陽

呼び出し関係図:



5.1.1.12 ProcessAllTransforms()

void SampleScenes::ProcessAllTransforms (
World & world) [inline]

レベル 7: 全エンティティに対する処理

学べること:

- ForEach で全エンティティを巡回
- ラムダ式の使い方

全Transform を持つエンティティを少しずつ上に移動

引数

in,out	world	ワ

著者

山内陽

呼び出し関係図:

Chapter 6

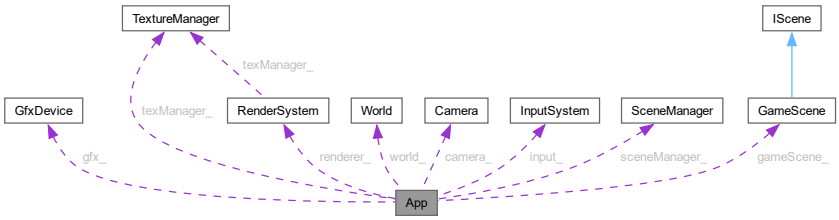
クラス詳解

6.1 App 構造体

ミニゲームのメインアプリケーションクラス

#include <App.h>

App 連携図



公開メンバ関数

- bool Init (HINSTANCE hInst, int width=1280, int height=720)
  アプリケーションの初期化
- void Run ()
  メインループの実行
- ~App ()
  デストラクタ

公開変数類

- HWND hwnd_ = nullptr
  メインウィンドウのハンドル
- GfxDevice gfx_
  グラフィックスデバイス
- RenderSystem renderer_
  描画システム
- TextureManager texManager_
  テクスチャ管理
- World world_
  ECS ワールド
- Camera camera_
  カメラ
- InputSystem input_
  入力システム
- SceneManager sceneManager_
  シーンマネージャー
- GameScene * gameScene_ = nullptr
  現在のゲームシーン

6.1.1 詳解

ミニゲームのメインアプリケーションクラス

ウィンドウ作成、DirectX11 の初期化、ECS ワールドの管理、メインループの実行など、アプリケーション全体のライフサイクルを管理します。

6.1.2 構築子と解体子

6.1.2.1 ~App()

App::~App () [inline]

デストラクタ

6.1.3 関数詳解

6.1.3.1 Init()

bool App::Init (
HINSTANCE hInst,
int width = 1280,
int height = 720) [inline]

アプリケーションの初期化

引数

in	hInst	アプリ
in	width	
in	height	

戻り値

bool 初期化が成功した場合は true, それ以外は false

6.1.3.2 Run()

void App::Run () [inline]

メインループの実行

アプリケーションが終了するまで、メッセージ処理、更新、描画を繰り返します。

6.1.4 メンバ詳解

6.1.4.1 camera_

Camera App::camera_

カメラ

6.1.4.2 gameScene_

GameScene* App::gameScene_ = nullptr

現在のゲームシーン

6.1.4.3 gfx_

GfxDevice App::gfx_

グラフィックスデバイス

6.1.4.4 hwnd_

HWND App::hwnd_ = nullptr

メインウィンドウのハンドル

6.1.4.5 input_

InputSystem App::input_

入力システム

6.1.4.6 renderer_

RenderSystem App::renderer_

描画システム

6.1.4.7 sceneManager_

SceneManager App::sceneManager_

シーンマネージャー

6.1.4.8 texManager_

TextureManager App::texManager_

テクスチャ管理

6.1.4.9 world_

World App::world_

ECS ワールド

この構造体詳解は次のファイルから抽出されました:

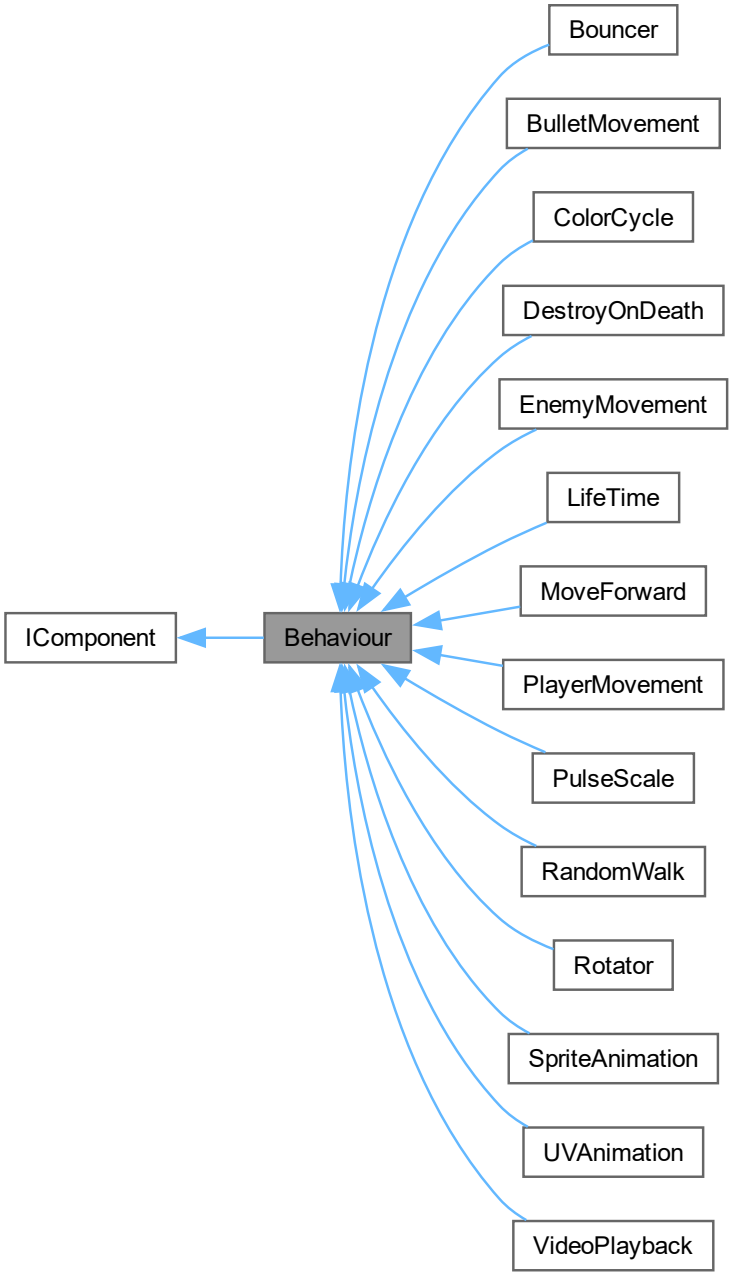
- include/app/App.h

6.2 Behaviour クラス

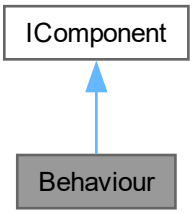
毎フレーム更新される動的コンポーネントの基底クラス

#include <Component.h>

Behaviour の継承関係図



Behaviour 連携図



公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド
- virtual void OnUpdate (World &w, Entity self, float dt)
  毎フレーム呼ばれる更新メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.2.1 詳解

毎フレーム更新される動的コンポーネントの基底クラス

このクラスを継承することで、ゲームロジックを持つコンポーネントを作成できます。OnUpdate() メソッドが毎フレーム自動的に呼ばれます。

6.2.1.0.1 用途:

- オブジェクトを動かす（移動、回転、拡大縮小）
- 時間経過で何かを変化させる
- アニメーションを再生する
- ゲームロジックを実行する

6.2.1.0.2 **ライフサイクル**:

1. OnStart() - エンティティが作成された直後に 1 度だけ呼ばれる（初期化用）
2. OnUpdate() - 毎フレーム呼ばれる（dt = 前フレームからの経過時間）

使用例:

```
// 自動回転コンポーネント
struct Rotator : Behaviour {
    float speed = 45.0f;  // 毎秒 45 度回転

    void OnUpdate(World& w, Entity self, float dt) override {
        auto* transform = w.TryGet<Transform>(self);
        if (transform) {
            transform->rotation.y += speed * dt;
        }
    }
};
```

参照

IComponent 基底インターフェース
World コンポーネントを管理するクラス

著者

山内陽

6.2.2 関数詳解

6.2.2.1 OnStart()

virtual void Behaviour::OnStart (
World & w,
Entity self) [inline], [virtual]

エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

引数

in	w	ワールド
in	self	

初期化処理が必要な場合にオーバーライドします。例: 初期位置の設定、初期状態の計算など

覚え書き

デフォルト実装は何もしません

使用例:

```
struct MyBehaviour : Behaviour {
    void OnStart(World& w, Entity self) override {
        // 初期化処理
        auto* transform = w.TryGet<Transform>(self);
        if (transform) {
            transform->position.y = 5.0f;  // 初期位置を設定
        }
    }
};
```

Bouncer, RandomWalk, VideoPlaybackで再実装されています。

6.2.2.2 OnUpdate()

```
virtual void Behaviour::OnUpdate (
                World & w,
                Entity self,
                float dt)   [inline], [virtual]
```

毎フレーム呼ばれる更新メソッド

引数

in,out	w	ワー
in	self	
in	dt	

ゲームロジックを実装する際にオーバーライドします。dt を使うことで、フレームレートに依存しない処理を実現できます。

覚え書き

デフォルト実装は何もしません

使用例:

```
struct MoveForward : Behaviour {
    float speed = 5.0f;

    void OnUpdate(World& w, Entity self, float dt) override {
        auto* transform = w.TryGet<Transform>(self);
        if (transform) {
            // dt を使ってフレームレート非依存な移動
            transform->position.z += speed * dt;
        }
    }
};
```

Bouncer, BulletMovement, ColorCycle, DestroyOnDeath, EnemyMovement, LifeTime, MoveForward, PlayerMovement, PulseScale, RandomWalk, Rotator, SpriteAnimation, UVAnimation, VideoPlaybackで再実装されています。

このクラス詳解は次のファイルから抽出されました:

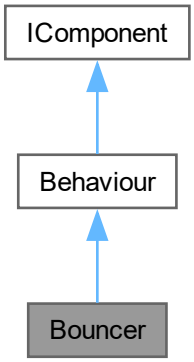
- include/components/Component.h

6.3 Bouncer 構造体

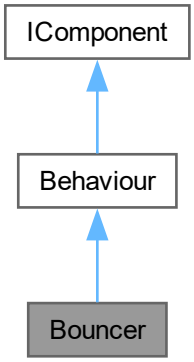
シンプルなBehaviour

#include <ComponentSamples.h>

Bouncer の継承関係図



Bouncer 連携図



公開メンバ関数

- void OnStart (World &w, Entity self) override
  初期化処理
- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speed = 2.0f
  跳ねる速度
- float amplitude = 2.0f
  振幅
- float time = 0.0f
  経過時間（内部管理）
- float startY = 0.0f
  開始位置（内部管理）

6.3.1 詳解

シンプルなBehaviour

1 つの明確な動作を実装学習ポイント：OnUpdate の使い方

上下に跳ねる（バウンス）Behaviour

sin 波を使ってエンティティを上下に跳ねさせます。OnStart で初期位置を記録し、OnUpdate で位置を更新します。

著者

山内陽

6.3.2 関数詳解

6.3.2.1 OnStart()

```
void Bouncer::OnStart (
            World & w,
            Entity self)    [inline], [override], [virtual]
```

初期化処理

引数

in,out	w	
in	self	自身

Behaviourを再実装しています。

呼び出し関係図:

6.3.2.2 OnUpdate()

void Bouncer::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:

6.3.3 メンバ詳解

6.3.3.1 amplitude

float Bouncer::amplitude = 2.0f

振幅

6.3.3.2 speed

float Bouncer::speed = 2.0f

跳ねる速度

6.3.3.3 startY

float Bouncer::startY = 0.0f

開始位置（内部管理）

6.3.3.4 time

float Bouncer::time = 0.0f

経過時間（内部管理）

この構造体詳解は次のファイルから抽出されました:

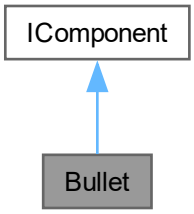
- include/samples/ComponentSamples.h

6.4 Bullet 構造体

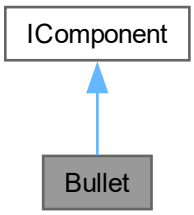
弾タグ

#include <MiniGame.h>

Bullet の継承関係図



Bullet 連携図



その他の継承メンバ

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.4.1 詳解

弾タグ

弾エンティティを識別するためのマーカー

この構造体詳解は次のファイルから抽出されました:

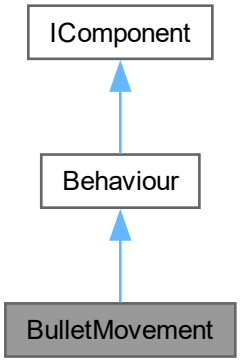
- include/scenes/MiniGame.h

6.5 BulletMovement 構造体

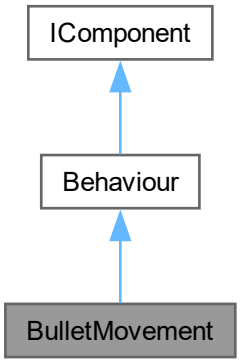
弾の移動Behaviour

#include <MiniGame.h>

BulletMovement の継承関係図



BulletMovement 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speed = 15.0f
  移動速度

6.5.1 詳解

弾の移動Behaviour

弾を上方向に移動させ、画面外に出たら削除します。

著者

山内陽

6.5.2 関数詳解

6.5.2.1 OnUpdate()

void BulletMovement::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

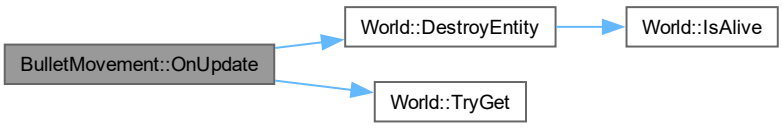
毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:



6.5.3 メンバ詳解

6.5.3.1 speed

float BulletMovement::speed = 15.0f

移動速度

この構造体詳解は次のファイルから抽出されました:

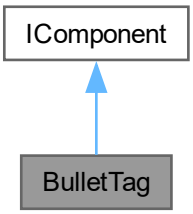
- include/scenes/MiniGame.h

6.6 BulletTag 構造体

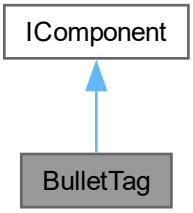
弾タグ

#include <ComponentSamples.h>

BulletTag の継承関係図



BulletTag 連携図



その他の継承メンバ

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.6.1 詳解

弾タグ

この構造体詳解は次のファイルから抽出されました:

- include/samples/ComponentSamples.h

6.7 Camera 構造体

3D 空間のカメラ（ビュー・プロジェクション行列）を管理

#include <Camera.h>

公開メンバ関数

- void Update ()
  カメラの更新（位置や注視点を変更した後に呼ぶ）
- void Orbit (float deltaYaw, float deltaPitch)
  オービットカメラ（ターゲットの周りを回転）
- void Zoom (float delta)
  ズーム（視野角の変更）

静的公開メンバ関数

- static Camera LookAtLH (float fovY, float aspect, float znear, float zfar, DirectX::XMFLOAT3 eye, DirectX::XMFLOAT3 at, DirectX::XMFLOAT3 upVec)
  LookAtLH カメラの作成（左手座標系）

公開変数類

- DirectX::XMMATRIX View
  ビュー行列（カメラの位置・向き）
- DirectX::XMMATRIX Proj
  プロジェクション行列（投影方法）
- DirectX::XMFLOAT3 position
  カメラの位置
- DirectX::XMFLOAT3 target
  カメラが見ている点（注視点）
- DirectX::XMFLOAT3 up
  カメラの上方向ベクトル
- float fovY
  垂直視野角（ラジアン）
- float aspect
  アスペクト比（幅/高さ）
- float nearZ
  ニアクリッププレーン
- float farZ
  ファークリッププレーン

6.7.1 詳解

3D 空間のカメラ（ビュー・プロジェクション行列）を管理

カメラは「どこから」「どこを」「どの向きで」見るかを制御します。また、レンダリングに必要なビュー行列とプロジェクション行列を保持します。

6.7.1.0.1 **カメラの構成要素**:

- View **行列**: カメラの位置と向きを表す
- Projection **行列**: 3D 空間を 2D 画面に投影する方法を表す

6.7.1.0.2 **座標系**:

- X 軸: 右方向
- Y 軸: 上方向
- Z 軸: 奥方向（カメラの視線方向）

使用例（基本）:

```
// カメラを作成（座標 (0,2,-5) から原点を見る）
Camera camera = Camera::LookAtLH(
    DirectX::XM_PIDIV4,              // 視野角 45 度
    1280.0f / 720.0f,                // アスペクト比
    0.1f,                            // ニアクリップ
    1000.0f,                         // ファークリップ
    DirectX::XMFLOAT3{0, 2, -5},     // カメラ位置
    DirectX::XMFLOAT3{0, 0, 0},      // 注視点
    DirectX::XMFLOAT3{0, 1, 0}       // 上方向
);
```

使用例（オービット回転）:

```
// カメラを回転（マウスドラッグ等）
camera.Orbit(deltaYaw, deltaPitch);  // 左右・上下回転
camera.Zoom(-0.1f);                  // ズームイン
```

参照

DirectX::XMMATRIX 行列型

著者

山内陽

6.7.2 関数詳解

6.7.2.1 LookAtLH()

Camera Camera::LookAtLH (
float fovY,
float aspect,
float znear,
float zfar,
DirectX::XMFLOAT3 eye,
DirectX::XMFLOAT3 at,
DirectX::XMFLOAT3 upVec) [inline], [static]

LookAtLH カメラの作成（左手座標系）

引数

in	fovY	垂直視
in	aspect	
in	znear	
in	zfar	
in	eye	
in	at	
in	upVec	

戻り値

Camera 設定されたカメラ

カメラを作成し、ビュー行列とプロジェクション行列を計算します。

視野角について:

- XM_PIDIV4 (/4) = 45 度: 標準的な視野角
- XM_PIDIV3 (/3) = 60 度: 広角
- XM_PIDIV6 (/6) = 30 度: 望遠

使用例:

```
Camera camera = Camera::LookAtLH(
    DirectX::XM_PIDIV4,              // 45 度
    1920.0f / 1080.0f,               // Full HD
    0.1f,                          // 0.1m 手前まで
    1000.0f,                        // 1000m 奥まで
    DirectX::XMFLOAT3{0, 5, -10},    // 少し後ろから
    DirectX::XMFLOAT3{0, 0, 0},      // 原点を見る
    DirectX::XMFLOAT3{0, 1, 0}       // Y 軸が上
);
```

6.7.2.2 Orbit()

```
void Camera::Orbit (
                float deltaYaw,
                float deltaPitch)   [inline]
```

オービットカメラ（ターゲットの周りを回転）

引数

in	deltaYaw	左
in	deltaPitch	上

注視点（target）を中心に、カメラを回転させます。マウスドラッグでカメラを回す際などに使用します。

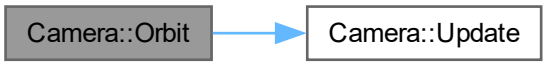
覚え書き

内部でUpdate() を呼ぶため、手動でUpdate() を呼ぶ必要はありません

使用例:

```
// マウスの移動量をカメラ回転に変換
float yaw = mouseDeltaX * 0.01f;    // 左右
float pitch = mouseDeltaY * 0.01f;  // 上下
camera.Orbit(yaw, pitch);
```

呼び出し関係図:



6.7.2.3 Update()

```
void Camera::Update ()   [inline]
```

カメラの更新（位置や注視点を変更した後に呼ぶ）

position、target、up を変更した後、この関数を呼ぶことでビュー行列が再計算されます。

使用例:

```
camera.position.y += 1.0f;  // カメラを上に移動
camera.Update();            // 行列を再計算
```

6.7.2.4 Zoom()

void Camera::Zoom (
float delta) [inline]

ズーム（視野角の変更）

引数

in	delta	視野角

視野角を変更することでズーム効果を実現します。視野角は 22.5 度〜90 度の範囲に制限されます。

使用例:

```
camera.Zoom(-0.1f);  // ズームイン（視野角を狭める）
camera.Zoom(0.1f);   // ズームアウト（視野角を広げる）
```

6.7.3 メンバ詳解

6.7.3.1 aspect

float Camera::aspect

アスペクト比（幅/高さ）

6.7.3.2 farZ

float Camera::farZ

ファークリッププレーン

6.7.3.3 fovY

float Camera::fovY

垂直視野角（ラジアン）

6.7.3.4 nearZ

float Camera::nearZ

ニアクリッププレーン

6.7.3.5 position

DirectX::XMFLOAT3 Camera::position

カメラの位置

6.7.3.6 Proj

DirectX::XMMATRIX Camera::Proj

プロジェクション行列（投影方法）

6.7.3.7 target

DirectX::XMFLOAT3 Camera::target

カメラが見ている点（注視点）

6.7.3.8 up

DirectX::XMFLOAT3 Camera::up

カメラの上方向ベクトル

6.7.3.9 View

DirectX::XMMATRIX Camera::View

ビュー行列（カメラの位置・向き）

この構造体詳解は次のファイルから抽出されました:

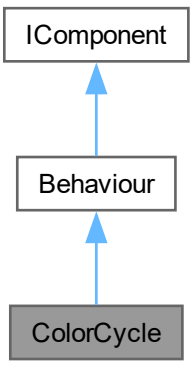
- include/graphics/Camera.h

6.8 ColorCycle 構造体

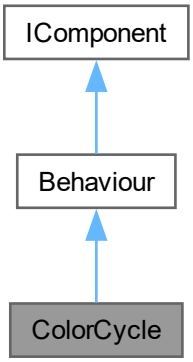
色を変化（サイクル）Behaviour

#include <ComponentSamples.h>

ColorCycle の継承関係図



ColorCycle 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speed = 1.0f
  色変化の速度
- float time = 0.0f
  経過時間（内部管理）

6.8.1 詳解

色を変化（サイクル）Behaviour

時間経過で色相を変化させ、虹色にサイクルします。

著者

山内陽

6.8.2 関数詳解

6.8.2.1 OnUpdate()

void ColorCycle::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

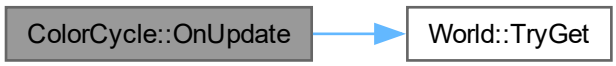
毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:



6.8.3 メンバ詳解

6.8.3.1 speed

float ColorCycle::speed = 1.0f

色変化の速度

6.8.3.2 time

float ColorCycle::time = 0.0f

経過時間（内部管理）

この構造体詳解は次のファイルから抽出されました:

- include/samples/ComponentSamples.h

6.9 DebugDraw クラス

デバッグ用の線描画システム

#include <DebugDraw.h>

クラス

- struct Line
  線分の定義（開始点、終了点、色）

公開メンバ関数

- bool Init (GfxDevice &gfx)
  初期化
- void AddLine (const DirectX::XMFLOAT3 &start, const DirectX::XMFLOAT3 &end, const DirectX::XMFLOAT3 &color)
  線を追加
- void DrawGrid (float size=10.0f, int divisions=10, const DirectX::XMFLOAT3 &color={0.5f, 0.5f, 0.5f})
  グリッドを描画
- void DrawAxes (float length=5.0f)
  座標軸を描画
- void Render (GfxDevice &gfx, const Camera &cam)
  すべての線を描画
- void Clear ()
  フレーム終了時にクリア
- ~DebugDraw ()
  デストラクタ

6.9.1 詳解

デバッグ用の線描画システム

開発中にグリッド、座標軸、任意の線を描画するためのクラスです。ワールド空間でのデバッグ情報の可視化に使用します。

6.9.1.0.1 主な用途:

- グリッド表示（基準となる平面）
- 座標軸表示（X, Y, Z 軸）
- 当たり判定の可視化
- 移動経路の表示

使用例:

```
DebugDraw debugDraw;
debugDraw.Init(gfx);

// グリッドと軸を描画
debugDraw.DrawGrid(20.0f, 20);
debugDraw.DrawAxes(5.0f);

// カスタム線を描画
debugDraw.AddLine(
    DirectX::XMFLOAT3{0, 0, 0},
    DirectX::XMFLOAT3{5, 5, 5},
    DirectX::XMFLOAT3{1, 1, 0}  // 黄色
);

// 描画実行
debugDraw.Render(gfx, camera);

// フレーム終了時にクリア
debugDraw.Clear();
```

覚え書き

デバッグビルド (_DEBUG 定義時) のみ使用を推奨

著者

山内陽

6.9.2 構築子と解体子

6.9.2.1 ~DebugDraw()

DebugDraw::~DebugDraw () [inline]

デストラクタ

6.9.3 関数詳解

6.9.3.1 AddLine()

void DebugDraw::AddLine (
const DirectX::XMFLOAT3 & start,
const DirectX::XMFLOAT3 & end,
const DirectX::XMFLOAT3 & color) [inline]

線を追加

引数

in	start	
in	end	
in	color	線の色

6.9.3.2 Clear()

void DebugDraw::Clear () [inline]

フレーム終了時にクリア

蓄積された線データをクリアします。毎フレーム呼び出す必要があります。

6.9.3.3 DrawAxes()

void DebugDraw::DrawAxes (
float length = 5.0f) [inline]

座標軸を描画

引数

in	length	軸の長

X 軸（赤）、Y 軸（緑）、Z 軸（青）を原点から描画します。

6.9.3.4 DrawGrid()

void DebugDraw::DrawGrid (
float size = 10.0f,
int divisions = 10,
const DirectX::XMFLOAT3 & color = {0.5f, 0.5f, 0.5f}) [inline]

グリッドを描画

引数

in	size	グ
in	divisions	
in	color	

6.9.3.5 Init()

bool DebugDraw::Init (
GfxDevice & gfx) [inline]

初期化

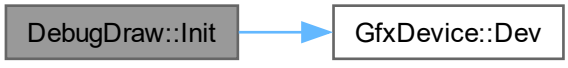
引数

in	gfx	グラフィ

戻り値

bool 初期化が成功した場合は true

呼び出し関係図:



6.9.3.6 Render()

void DebugDraw::Render (
GfxDevice & gfx,
const Camera & cam) [inline]

すべての線を描画

引数

in	gfx	グラフ
in	cam	

このクラス詳解は次のファイルから抽出されました:

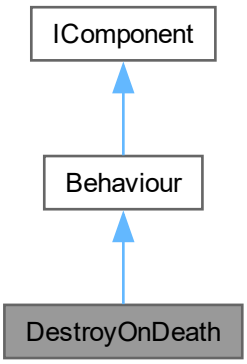
- include/graphics/DebugDraw.h

6.10 DestroyOnDeath 構造体

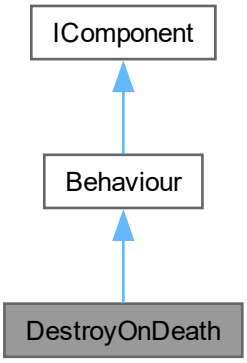
複雑なBehaviour

#include <ComponentSamples.h>

DestroyOnDeath の継承関係図



DestroyOnDeath 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.10.1 詳解

複雑なBehaviour

複数のコンポーネントを組み合わせる学習ポイント：コンポーネント間の連携

体力が 0 になったら削除するBehaviour

Health コンポーネントを監視し、体力が 0 以下になったらエンティティを削除します。

著者

山内陽

6.10.2 関数詳解

6.10.2.1 OnUpdate()

void DestroyOnDeath::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

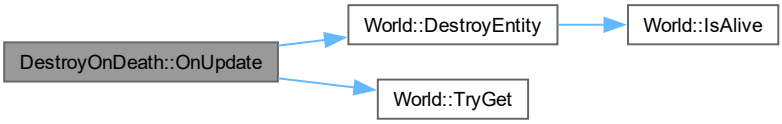
毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:



この構造体詳解は次のファイルから抽出されました:

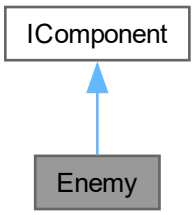
- include/samples/ComponentSamples.h

6.11 Enemy 構造体

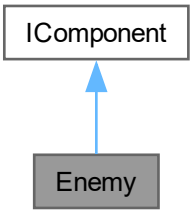
敵タグ

#include <MiniGame.h>

Enemy の継承関係図



Enemy 連携図



その他の継承メンバ

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.11.1 詳解

敵タグ

敵エンティティを識別するためのマーカー

この構造体詳解は次のファイルから抽出されました:

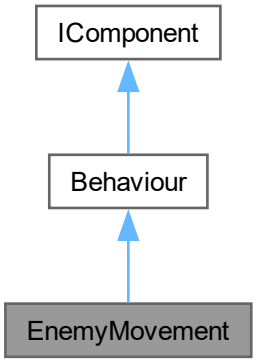
- include/scenes/MiniGame.h

6.12 EnemyMovement 構造体

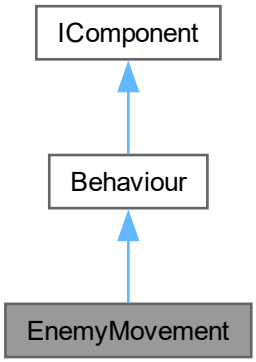
敵の移動Behaviour

#include <MiniGame.h>

EnemyMovement の継承関係図



EnemyMovement 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speed = 3.0f
  移動速度

6.12.1 詳解

敵の移動Behaviour

敵を下方向に移動させ、画面外に出たら削除します。

著者

山内陽

6.12.2 関数詳解

6.12.2.1 OnUpdate()

void EnemyMovement::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

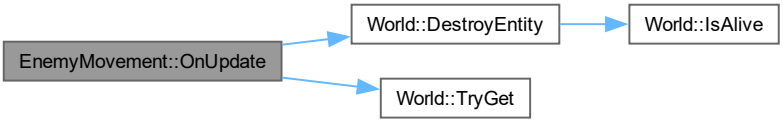
毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:



6.12.3 メンバ詳解

6.12.3.1 speed

float EnemyMovement::speed = 3.0f

移動速度

この構造体詳解は次のファイルから抽出されました:

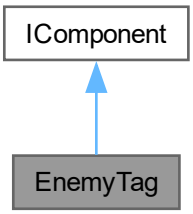
- include/scenes/MiniGame.h

6.13 EnemyTag 構造体

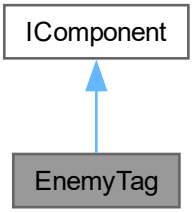
敵タグ

#include <ComponentSamples.h>

EnemyTag の継承関係図



EnemyTag 連携図



その他の継承メンバ

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.13.1 詳解

敵タグ

この構造体詳解は次のファイルから抽出されました:

- include/samples/ComponentSamples.h

6.14 Entity 構造体

ゲーム世界に存在するオブジェクトを表す一意な識別子

#include <Entity.h>

公開変数類

- uint32_t id
  エンティティの一意識別番号

6.14.1 詳解

ゲーム世界に存在するオブジェクトを表す一意な識別子

ECS アーキテクチャにおけるエンティティは、ゲーム世界の「物」を表す単なるID 番号です。エンティティ自体には機能がなく、コンポーネントを組み合わせることで機能を持たせます。

6.14.1.0.1 エンティティの特徴:

- **軽量**: uint32_t 型のID 番号のみを保持
- **一意**: World によってユニークなID が割り当てられる
- **柔軟**: コンポーネントの組み合わせで様々な機能を実現

6.14.1.0.2 **コンポーネント指向の考え方**:

従来のオブジェクト指向では、継承によって機能を追加しますが、ECS ではエンティティにコンポーネントを追加することで機能を実現します。

従来のオブジェクト指向（継承）:

```
// ダメな例: 継承による機能追加
class Player : public Character {
    // プレイヤー専用の処理をごちゃ混ぜ
};
```

ECS（コンポーネント）:

```
// 良い例: コンポーネントによる機能追加
Entity player = world.Create()
    .With<Transform>()      // 位置の機能
    .With<MeshRenderer>()   // 見た目の機能
    .With<PlayerInput>()    // 入力の機能
    .With<Health>()         // 体力の機能
    .Build();
```

6.14.1.0.3 **具体例**:

```
// プレイヤーエンティティの例
Entity player = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{0, 1, 0})  // 緑色
    .With<Player>()  // プレイヤータグ
    .Build();

// 敵エンティティの例
Entity enemy = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{5, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{1, 0, 0})  // 赤色
    .With<Enemy>()   // 敵タグ
    .With<Health>()  // 体力
    .Build();

// 弾エンティティの例
Entity bullet = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{1, 1, 0})  // 黄色
    .With<Bullet>()  // 弾タグ
    .Build();
```

覚え書き

エンティティのID は自動的に割り当てられるため、直接操作する必要はありません

警告

エンティティを削除する際は、必ずWorld::DestroyEntity() を使用してください

参照

World エンティティとコンポーネントを管理するクラス

IComponent コンポーネントの基底クラス

Transform 位置・回転・スケールコンポーネント

著者

山内陽

6.14.2 メンバ詳解

6.14.2.1 id

uint32_t Entity::id

エンティティの一意識別番号

World クラスによって自動的に割り当てられる一意なID 番号です。このID を使って、World からコンポーネントを取得・追加・削除します。

覚え書き

ID 0 は無効なエンティティを表す特殊値として使用される場合があります

警告

このID を直接変更しないでください

使用例:

```
Entity entity = world.CreateEntity();
std::cout « "Entity ID: " « entity.id « std::endl;

// ID を使ってコンポーネントを取得
auto* transform = world.TryGet<Transform>(entity);
```

この構造体詳解は次のファイルから抽出されました:

- include/ecs/Entity.h

6.15 EntityBuilder クラス

前方宣言

#include <World.h>

公開メンバ関数

- EntityBuilder (World *world, Entity entity)
  コンストラクタ
- template<typename T, typename... Args>
  EntityBuilder & With (Args &&... args)
  コンポーネントを追加する（メソッドチェーン対応）
- Entity Build ()
  エンティティを確定して返す
- operator Entity () const
  Entity への暗黙的型変換演算子

6.15.1 詳解

前方宣言

エンティティ作成を簡単にするビルダーパターンクラス

メソッドチェーンを使って、複数のコンポーネントを持つエンティティを簡潔に作成できます。World クラスと連携して動作します。

使用例:

```
Entity player = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{0, 1, 0})  // 緑色
    .With<Rotator>(45.0f)
    .Build();
```

覚え書き

Build()は省略可能です（暗黙的にEntity に変換されます）

参照

World エンティティとコンポーネントを管理するクラス

著者

山内陽

6.15.2 構築子と解体子

6.15.2.1 EntityBuilder()

```
EntityBuilder::EntityBuilder (
            World * world,
            Entity entity)  [inline]
```

コンストラクタ

引数

in	world	World
in	entity	作成さ

6.15.3 関数詳解

6.15.3.1 Build()

```
Entity EntityBuilder::Build ()  [inline]
```

エンティティを確定して返す

戻り値

Entity 作成されたエンティティ

覚え書き

この関数を呼ばなくても、暗黙的にEntity に変換されます

6.15.3.2 operator Entity()

EntityBuilder::operator Entity () const [inline]

Entity への暗黙的型変換演算子

戻り値

Entity 作成されたエンティティ

Build()を呼ばずに、直接Entity 型の変数に代入できます

使用例:

```
// Build() なしで直接代入
Entity e = world.Create().With<Transform>();
```

6.15.3.3 With()

```
template<typename T, typename... Args>
EntityBuilder & EntityBuilder::With (
                Args &&... args)
```

コンポーネントを追加する（メソッドチェーン対応）

EntityBuilder::With()の実装

テンプレート引数

	T	追加するコンポーネントの型
	Args	コンストラクタ引数の型（可変長）

引数

in	args	コンポ

戻り値

EntityBuilder& メソッドチェーン用の自身への参照

使用例:

```
world.Create()
    .With<Transform>(pos, rot, scale)  // 3 つの引数を渡す
    .With<MeshRenderer>(color)         // 1 つの引数を渡す
    .With<Player>()                    // 引数なし
    .Build();
```

テンプレート引数

	T	追加するコンポーネントの型
	Args	コンストラクタ引数の型

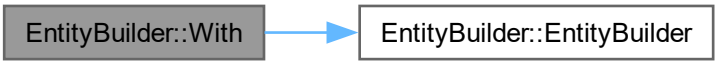
引数

in	args	コンポ

戻り値

EntityBuilder& メソッドチェーン用の自身への参照

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

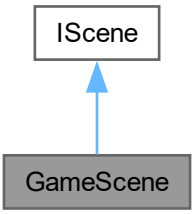
- include/ecs/World.h

6.16 GameScene クラス

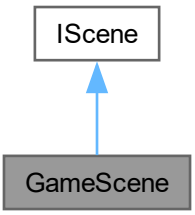
シューティングゲームのメインシーン

#include <MiniGame.h>

GameScene の継承関係図



GameScene 連携図



公開メンバ関数

- void OnEnter (World &world) override
  シーン開始時の初期化
- void OnUpdate (World &world, InputSystem &input, float deltaTime) override
  毎フレーム更新処理
- void OnExit (World &world) override
  シーン終了時のクリーンアップ
- int GetScore () const
  現在のスコアを取得

基底クラス IScene に属する継承公開メンバ関数

- virtual ~IScene ()=default
  仮想デストラクタ
- virtual bool ShouldChangeScene () const
  次のシーンへ遷移するか判定
- virtual const char * GetNextScene () const
  次のシーン名を取得

6.16.1 詳解

シューティングゲームのメインシーン

プレイヤー操作、敵の生成、弾の発射、衝突判定を管理します。

6.16.1.0.1 ゲームルール:

- A/D キーでプレイヤーを左右に移動
- スペースキーで弾を発射
- 敵が上から降ってくる
- 弾が敵に当たると両方消滅し、スコア +10

著者

山内陽

6.16.2 関数詳解

6.16.2.1 GetScore()

int GameScene::GetScore () const [inline]

現在のスコアを取得

戻り値

int 現在のスコア

6.16.2.2 OnEnter()

void GameScene::OnEnter (
World & world) [inline], [override], [virtual]

シーン開始時の初期化

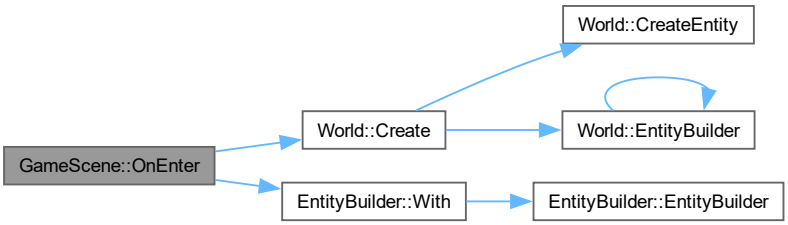
引数

in,out	world	ワ

プレイヤーを生成し、スコアとタイマーを初期化します。

IScene を実装しています。

呼び出し関係図:



6.16.2.3 OnExit()

void GameScene::OnExit (
World & world) [inline], [override], [virtual]

シーン終了時のクリーンアップ

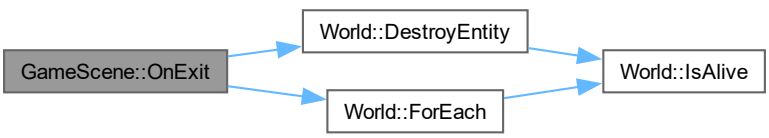
引数

in,out	world	ワ

すべてのエンティティを削除します。

IScene を実装しています。

呼び出し関係図:



6.16.2.4 OnUpdate()

void GameScene::OnUpdate (
World & world,
InputSystem & input,
float deltaTime) [inline], [override], [virtual]

毎フレーム更新処理

引数

in,out	world
in	input
in	deltaTime

IScene を実装しています。

呼び出し関係図:

このクラス詳解は次のファイルから抽出されました:

- include/scenes/MiniGame.h

6.17 GfxDevice クラス

DirectX11 デバイス管理クラス

#include <GfxDevice.h>

公開メンバ関数

- bool Init (HWND hwnd, uint32_t w, uint32_t h)
  初期化
- void BeginFrame (float r=0.1f, float g=0.1f, float b=0.12f, float a=1.0f)
  フレーム開始（画面クリア）
- void EndFrame ()
  フレーム終了（画面表示）
- ID3D11Device * Dev () const
  デバイスアクセス
- ID3D11DeviceContext * Ctx () const
  デバイスコンテキストアクセス
- uint32_t Width () const
  幅を取得
- uint32_t Height () const
  高さを取得
- ~GfxDevice ()
  デストラクタでリソースを明示的に解放

6.17.1 詳解

DirectX11 デバイス管理クラス

DirectX11 のデバイス、スワップチェイン、レンダーターゲット、深度バッファなどを管理し、描画フレームの開始・終了を制御します。

6.17.2 構築子と解体子

6.17.2.1 ~GfxDevice()

GfxDevice::~GfxDevice () [inline]

デストラクタでリソースを明示的に解放

6.17.3 関数詳解

6.17.3.1 BeginFrame()

void GfxDevice::BeginFrame (
float r = 0.1f,
float g = 0.1f,
float b = 0.12f,
float a = 1.0f) [inline]

フレーム開始（画面クリア）

引数

in	r	赤成
in	g	緑成
in	b	青成
in	a	アルファ成

6.17.3.2 Ctx()

ID3D11DeviceContext * GfxDevice::Ctx () const [inline]

デバイスコンテキストアクセス

戻り値

ID3D11DeviceContext* デバイスコンテキストポインタ

6.17.3.3 Dev()

ID3D11Device * GfxDevice::Dev () const [inline]

デバイスアクセス

戻り値

ID3D11Device* デバイスポインタ

6.17.3.4 EndFrame()

void GfxDevice::EndFrame () [inline]

フレーム終了（画面表示）

6.17.3.5 Height()

uint32_t GfxDevice::Height () const [inline]

高さを取得

戻り値

uint32_t 高さ

6.17.3.6 Init()

bool GfxDevice::Init (
HWND hwnd,
uint32_t w,
uint32_t h) [inline]

初期化

引数

in	hwnd	ウィン
in	w	
in	h	

戻り値

bool 初期化が成功した場合は true

6.17.3.7 Width()

uint32_t GfxDevice::Width () const [inline]

幅を取得

戻り値

uint32_t 幅

このクラス詳解は次のファイルから抽出されました:

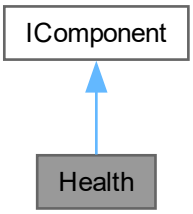
- include/graphics/GfxDevice.h

6.18 Health 構造体

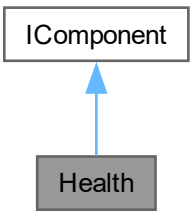
データ型のコンポーネント

#include <ComponentSamples.h>

Health の継承関係図



Health 連携図



公開メンバ関数

- void TakeDamage (float damage)
  ダメージを受ける
- void Heal (float amount)
  回復する
- bool IsDead () const
  死亡しているか

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float current = 100.0f
  現在の体力
- float max = 100.0f
  最大体力

6.18.1 詳解

データ型のコンポーネント

状態を保存するだけで、動作はしない使い方：他のBehaviour から参照される

体力コンポーネント

エンティティの体力を管理します。ダメージや回復、死亡判定を提供します。

著者

山内陽

6.18.2 関数詳解

6.18.2.1 Heal()

void Health::Heal (
float amount) [inline]

回復する

引数

in	amount	回復

6.18.2.2 IsDead()

bool Health::IsDead () const [inline]

死亡しているか

戻り値

true 死亡している, false 生存している

6.18.2.3 TakeDamage()

void Health::TakeDamage (
float damage) [inline]

ダメージを受ける

引数

in	damage	ダメ

6.18.3 メンバ詳解

6.18.3.1 current

float Health::current = 100.0f

現在の体力

6.18.3.2 max

float Health::max = 100.0f

最大体力

この構造体詳解は次のファイルから抽出されました:

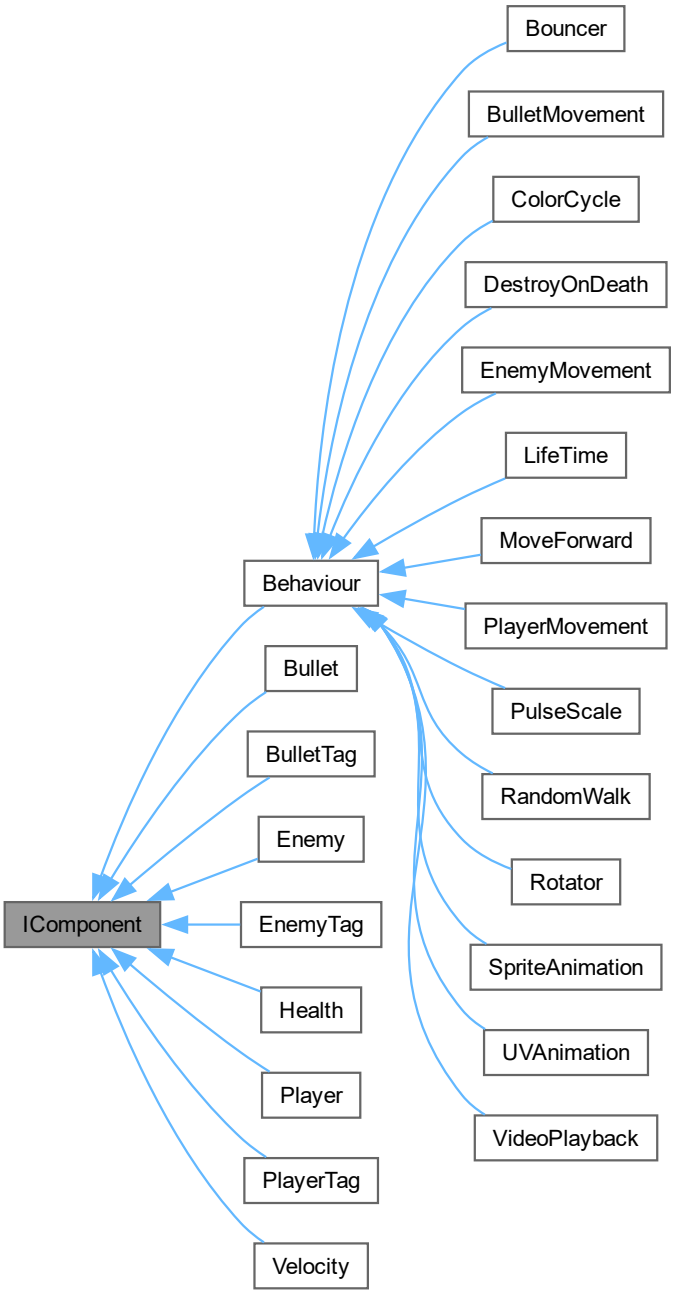
- include/samples/ComponentSamples.h

6.19 IComponent インタフェース

前方宣言: Entity 構造体

#include <Component.h>

IComponent の継承関係図



公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.19.1 詳解

前方宣言: Entity 構造体

すべてのコンポーネントの基底インターフェース

このクラスは、型情報を保持するための共通の親クラスです。実際のコンポーネントはこのクラスを継承して作成します。

覚え書き

初学者は特に意識する必要はありません。継承することで、World がコンポーネントを管理できるようになります。

使用例:

```
// データコンポーネントの例
struct Health : IComponent {
    float hp = 100.0f;
    float maxHp = 100.0f;
};
```

著者

山内陽

6.19.2 構築子と解体子

6.19.2.1 ~IComponent()

virtual IComponent::~IComponent () [virtual], [default]

仮想デストラクタ

ポリモーフィズムを可能にするため、仮想デストラクタを定義

このインタフェース詳解は次のファイルから抽出されました:

- include/components/Component.h

6.20 InputSystem クラス

キーボード・マウス入力を管理するクラス

#include <InputSystem.h>

公開型

- enum class KeyState : uint8_t { None = 0 , Down = 1 , Pressed = 2 , Up = 3 }
  キーの状態を表す列挙型
- enum MouseButton { Left = 0 , Right = 1 , Middle = 2 }
  マウスボタンの識別子

公開メンバ関数

- void Init ()
  初期化
- void Update ()
  入力状態の更新（毎フレーム呼ぶ）
- bool GetKey (int vkCode) const
  キーが押されているか
- bool GetKeyDown (int vkCode) const
  キーがこのフレームで押された瞬間か
- bool GetKeyUp (int vkCode) const
  キーがこのフレームで離された瞬間か
- bool GetMouseButton (MouseButton button) const
  マウスボタンが押されているか
- bool GetMouseButtonDown (MouseButton button) const
  マウスボタンがこのフレームで押された瞬間か
- bool GetMouseButtonUp (MouseButton button) const
  マウスボタンがこのフレームで離された瞬間か
- int GetMouseX () const
  マウスX 座標を取得
- int GetMouseY () const
  マウスY 座標を取得
- int GetMouseDeltaX () const
  マウスの移動量（X）を取得
- int GetMouseDeltaY () const
  マウスの移動量（Y）を取得
- int GetMouseWheel () const
  マウスホイールの回転量を取得
- void OnMouseWheel (int delta)
  マウスホイールイベント

6.20.1 詳解

キーボード・マウス入力を管理するクラス

Windows API を使用してキーボードとマウスの入力状態を管理します。ゲームループ内で毎フレームUpdate() を呼び出すことで、入力状態が更新されます。

使用例:

```
InputSystem input;
input.Init();

while (running) {
    input.Update();

    if (input.GetKeyDown(VK_SPACE)) {
        // スペースキー押下
    }
}
```

著者

山内陽

6.20.2 列挙型メンバ詳解

6.20.2.1 KeyState

enum class InputSystem::KeyState : uint8_t [strong]

キーの状態を表す列挙型

列挙値

None	何も押されていない
Down	このフレームで押された
Pressed	押され続けている
Up	このフレームで離された

6.20.2.2 MouseButton

enum InputSystem::MouseButton

マウスボタンの識別子

列挙値

Left	左ボタン
Right	右ボタン
Middle	中ボタン

6.20.3 関数詳解

6.20.3.1 GetKey()

bool InputSystem::GetKey (
int vkCode) const [inline]

キーが押されているか

引数

in	vkCode	仮想

戻り値

true 押されている, false 押されていない

6.20.3.2 GetKeyDown()

bool InputSystem::GetKeyDown (
int vkCode) const [inline]

キーがこのフレームで押された瞬間か

引数

	in	vkCode	仮想

戻り値

true 押された瞬間, false それ以外

6.20.3.3 GetKeyUp()

bool InputSystem::GetKeyUp (
int vkCode) const [inline]

キーがこのフレームで離された瞬間か

引数

	in	vkCode	仮想

戻り値

true 離された瞬間, false それ以外

6.20.3.4 GetMouseButton()

bool InputSystem::GetMouseButton (
MouseButton button) const [inline]

マウスボタンが押されているか

引数

	in	button	マウ

戻り値

true 押されている, false 押されていない

呼び出し関係図:

6.20.3.5 GetMouseButtonDown()

bool InputSystem::GetMouseButtonDown (
MouseButton button) const [inline]

マウスボタンがこのフレームで押された瞬間か

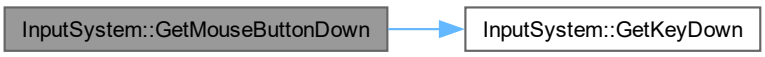
引数

in	button	マウ

戻り値

true 押された瞬間, false それ以外

呼び出し関係図:



6.20.3.6 GetMouseButtonUp()

bool InputSystem::GetMouseButtonUp (
MouseButton button) const [inline]

マウスボタンがこのフレームで離された瞬間か

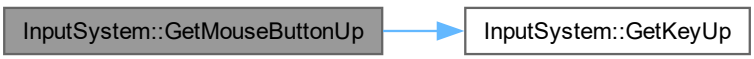
引数

in	button	マウ

戻り値

true 離された瞬間, false それ以外

呼び出し関係図:



6.20.3.7 GetMouseDeltaX()

int InputSystem::GetMouseDeltaX () const [inline]

マウスの移動量（X）を取得

戻り値

int X 方向移動量

6.20.3.8 GetMouseDeltaY()

int InputSystem::GetMouseDeltaY () const [inline]

マウスの移動量（Y）を取得

戻り値

int Y 方向移動量

6.20.3.9 GetMouseWheel()

int InputSystem::GetMouseWheel () const [inline]

マウスホイールの回転量を取得

戻り値

int ホイール回転量

6.20.3.10 GetMouseX()

int InputSystem::GetMouseX () const [inline]

マウスX 座標を取得

戻り値

int X 座標

6.20.3.11 GetMouseY()

int InputSystem::GetMouseY () const [inline]

マウスY 座標を取得

戻り値

int Y 座標

6.20.3.12 Init()

void InputSystem::Init () [inline]

初期化

6.20.3.13 OnMouseWheel()

void InputSystem::OnMouseWheel (
int delta) [inline]

マウスホイールイベント

引数

in	delta	ホイー

6.20.3.14 Update()

void InputSystem::Update () [inline]

入力状態の更新（毎フレーム呼ぶ）

このクラス詳解は次のファイルから抽出されました:

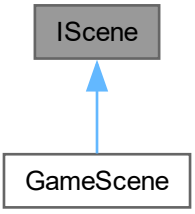
- include/input/InputSystem.h

6.21 IScene クラス

すべてのシーンの基底クラス

#include <SceneManager.h>

IScene の継承関係図



公開メンバ関数

- virtual ~IScene ()=default
  仮想デストラクタ
- virtual void OnEnter (World &world)=0
  シーン開始時の初期化処理
- virtual void OnUpdate (World &world, InputSystem &input, float deltaTime)=0
  毎フレームの更新処理
- virtual void OnExit (World &world)=0
  シーン終了時のクリーンアップ処理
- virtual bool ShouldChangeScene () const
  次のシーンへ遷移するか判定
- virtual const char * GetNextScene () const
  次のシーン名を取得

6.21.1 詳解

すべてのシーンの基底クラス

ゲームの各画面 (シーン) はこのクラスを継承して作成します。

6.21.1.0.1 実装が必要なメソッド:

- OnEnter(): シーンが始まるときに 1 回だけ呼ばれる
- OnUpdate(): 毎フレーム呼ばれる (ゲームロジック)
- OnExit(): シーンが終わるときに 1 回だけ呼ばれる

6.21.1.0.2 オプションのメソッド:

- ShouldChangeScene(): 次のシーンへ遷移するか判定
- GetNextScene(): 次のシーン名を返す

使用例:

```
class TitleScene : public IScene {
public:
    void OnEnter(World& world) override {
        // タイトル画面の初期化
    }

    void OnUpdate(World& world, InputSystem& input, float dt) override {
        // スペースキーでゲーム開始
        if (input.GetKeyDown(VK_SPACE)) {
            changeScene_ = true;
        }
    }

    void OnExit(World& world) override {
        // クリーンアップ
    }

    bool ShouldChangeScene() const override { return changeScene_; }
    const char* GetNextScene() const override { return "GameScene"; }

private:
    bool changeScene_ = false;
};
```

著者

山内陽

6.21.2 構築子と解体子

6.21.2.1 ~IScene()

virtual IScene::~IScene () [virtual], [default]

仮想デストラクタ

6.21.3 関数詳解

6.21.3.1 GetNextScene()

virtual const char * IScene::GetNextScene () const [inline], [virtual]

次のシーン名を取得

戻り値

const char* 次のシーン名

ShouldChangeScene()が true のときに呼ばれます。RegisterScene() で登録した名前を返してください。

使用例:

```
const char* GetNextScene() const override {
    return "ResultScene";
}
```

6.21.3.2 OnEnter()

virtual void IScene::OnEnter (
World & world) [pure virtual]

シーン開始時の初期化処理

引数

	in,out	world	ワ

シーンが開始されるときに 1 回だけ呼ばれます。エンティティの生成、変数の初期化などを行います。

使用例:

```
void OnEnter(World& world) override {
    // プレイヤーを生成
    player_ = world.Create()
        .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
        .With<MeshRenderer>(DirectX::XMFLOAT3{1, 0, 0})
        .Build();
}
```

GameSceneで実装されています。

6.21.3.3 OnExit()

virtual void IScene::OnExit (
World & world) [pure virtual]

シーン終了時のクリーンアップ処理

引数

in,out	world	ワ

シーンが終了するときに 1 回だけ呼ばれます。エンティティの削除、リソースの解放などを行います。

使用例:

```
void OnExit(World& world) override {
    // すべてのエンティティを削除
    world.DestroyEntity(player_);
    world.DestroyEntity(enemy_);
}
```

GameSceneで実装されています。

6.21.3.4 OnUpdate()

virtual void IScene::OnUpdate (
World & world,
InputSystem & input,
float deltaTime) [pure virtual]

毎フレームの更新処理

引数

in,out	world
in	input
in	deltaTime

毎フレーム呼ばれます。入力処理、移動、衝突判定などのゲームロジックを実装します。

使用例:

```
void OnUpdate(World& world, InputSystem& input, float dt) override {
    // 入力処理
    if (input.GetKey('W')) {
        // プレイヤーを移動
    }

    // ゲームロジック更新
    world.Tick(dt);
}
```

GameSceneで実装されています。

6.21.3.5 ShouldChangeScene()

virtual bool IScene::ShouldChangeScene () const [inline], [virtual]

次のシーンへ遷移するか判定

戻り値

true 遷移する, false 遷移しない

SceneManager が毎フレーム呼び出し、true ならシーンを切り替えます。

使用例:

```
bool ShouldChangeScene() const override {
    return gameOver_;  // ゲームオーバーなら遷移
}
```

このクラス詳解は次のファイルから抽出されました:

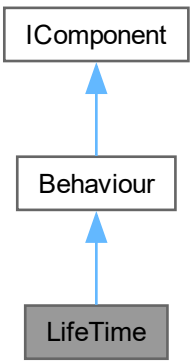
- include/scenes/SceneManager.h

6.22 LifeTime 構造体

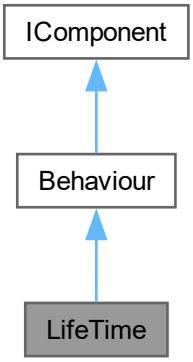
時間経過で削除するBehaviour

#include <ComponentSamples.h>

LifeTime の継承関係図



LifeTime 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float remainingTime = 5.0f
  残り時間（秒）

6.22.1 詳解

時間経過で削除するBehaviour

指定された時間が経過したらエンティティを削除します。一時的なエフェクトなどに使用します。

著者

山内陽

6.22.2 関数詳解

6.22.2.1 OnUpdate()

void LifeTime::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:

6.22.3 メンバ詳解

6.22.3.1 remainingTime

float LifeTime::remainingTime = 5.0f

残り時間（秒）

この構造体詳解は次のファイルから抽出されました:

- include/samples/ComponentSamples.h

6.23 DebugDraw::Line 構造体

線分の定義（開始点、終了点、色）

#include <DebugDraw.h>

公開変数類

- DirectX::XMFLOAT3 start
  線の開始点
- DirectX::XMFLOAT3 end
  線の終了点
- DirectX::XMFLOAT3 color
  線の色（RGB: 0.0〜1.0）

6.23.1 詳解

線分の定義（開始点、終了点、色）

6.23.2 メンバ詳解

6.23.2.1 color

DirectX::XMFLOAT3 DebugDraw::Line::color

線の色（RGB: 0.0〜1.0）

6.23.2.2 end

DirectX::XMFLOAT3 DebugDraw::Line::end

線の終了点

6.23.2.3 start

DirectX::XMFLOAT3 DebugDraw::Line::start

線の開始点

この構造体詳解は次のファイルから抽出されました:

- include/graphics/DebugDraw.h

6.24 MeshRenderer 構造体

オブジェクトの見た目（色・テクスチャ）を管理するデータコンポーネント

#include <MeshRenderer.h>

公開変数類

- DirectX::XMFLOAT3 color { 0.3f, 0.7f, 1.0f }
  オブジェクトの基本色（RGB: 0.0〜1.0）
- TextureManager::TextureHandle texture = TextureManager::INVALID_TEXTURE
  表面に貼り付けるテクスチャ画像のハンドル
- DirectX::XMFLOAT2 uvOffset { 0.0f, 0.0f }
  UV 座標のオフセット（テクスチャ位置のずらし）
- DirectX::XMFLOAT2 uvScale { 1.0f, 1.0f }
  UV 座標のスケール（テクスチャの繰り返し）

6.24.1 詳解

オブジェクトの見た目（色・テクスチャ）を管理するデータコンポーネント

このコンポーネントは 3D オブジェクトの描画設定を保持します。単色表示とテクスチャ表示の両方に対応しています。

6.24.1.0.1 **描画の仕組み**:

1. テクスチャが設定されていない場合: color で指定した単色で描画
2. テクスチャが設定されている場合: テクスチャ画像で描画（color は色調として使用）

6.24.1.0.2 UV **座標について**:

UV 座標は、テクスチャのどの部分を表示するかを指定します。

- uvOffset: テクスチャの開始位置をずらす（アニメーション等に使用）
- uvScale: テクスチャの繰り返し回数（タイリング）

使用例（単色）:

```
Entity cube = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{1, 0, 0})  // 赤色
    .Build();
```

使用例（テクスチャ）:

```
MeshRenderer renderer;
renderer.color = DirectX::XMFLOAT3{1, 1, 1};  // 白（テクスチャ本来の色）
renderer.texture = texManager.LoadFromFile("brick.png");

Entity cube = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(renderer)
    .Build();
```

使用例（UV アニメーション）:

```
// 毎フレーム、テクスチャを横にスクロール
auto* renderer = world.TryGet<MeshRenderer>(entity);
if (renderer) {
    renderer->uvOffset.x += 0.01f * dt;
}
```

覚え書き

Transform コンポーネントと組み合わせて使用します

参照

Transform 位置・回転・スケールを管理するコンポーネント
TextureManager テクスチャ管理クラス

著者

山内陽

6.24.2 メンバ詳解

6.24.2.1 color

DirectX::XMFLOAT3 MeshRenderer::color { 0.3f, 0.7f, 1.0f }

オブジェクトの基本色（RGB: 0.0〜1.0）

- テクスチャなし: この色で描画
- テクスチャあり: テクスチャにこの色を乗算（色調補正）

色の指定方法:

- color.x = R（赤）: 0.0〜1.0
- color.y = G（緑）: 0.0〜1.0
- color.z = B（青）: 0.0〜1.0

よく使う色:

```
DirectX::XMFLOAT3{1, 0, 0}  // 赤
DirectX::XMFLOAT3{0, 1, 0}  // 緑
DirectX::XMFLOAT3{0, 0, 1}  // 青
DirectX::XMFLOAT3{1, 1, 0}  // 黄
DirectX::XMFLOAT3{1, 0, 1}  // マゼンタ
DirectX::XMFLOAT3{0, 1, 1}  // シアン
DirectX::XMFLOAT3{1, 1, 1}  // 白
DirectX::XMFLOAT3{0, 0, 0}  // 黒
```

6.24.2.2 texture

TextureManager::TextureHandle MeshRenderer::texture = TextureManager::INVALID_TEXTURE

表面に貼り付けるテクスチャ画像のハンドル

TextureManager から取得したテクスチャハンドルを設定します。INVALID_TEXTURE の場合、テクスチャは使用されず単色で描画されます。

覚え書き

デフォルトはINVALID_TEXTURE（テクスチャなし）

使用例:

```
auto* renderer = world.TryGet<MeshRenderer>(entity);
if (renderer) {
    renderer->texture = texManager.LoadFromFile("brick.png");
}
```

参照

TextureManager::LoadFromFile テクスチャ読み込み

6.24.2.3 uvOffset

DirectX::XMFLOAT2 MeshRenderer::uvOffset { 0.0f, 0.0f }

UV 座標のオフセット（テクスチャ位置のずらし）

テクスチャの表示開始位置をずらします。アニメーションやスクロール効果に使用します。

- uvOffset.x: 横方向のオフセット（0.0〜1.0 で 1 周）
- uvOffset.y: 縦方向のオフセット（0.0〜1.0 で 1 周）

使用例（横スクロール）:

```
// 毎フレーム少しずつ横にずらす
renderer->uvOffset.x += 0.5f * dt;  // 毎秒 0.5 ずつスクロール
```

6.24.2.4 uvScale

DirectX::XMFLOAT2 MeshRenderer::uvScale { 1.0f, 1.0f }

UV 座標のスケール（テクスチャの繰り返し）

テクスチャの繰り返し回数を指定します（タイリング）。

- uvScale.x: 横方向の繰り返し回数
- uvScale.y: 縦方向の繰り返し回数

使用例（2x2 タイリング）:

```
renderer->uvScale = DirectX::XMFLOAT2{2.0f, 2.0f};  // 2x2 で繰り返し
```

覚え書き

1.0 が等倍（繰り返しなし）

この構造体詳解は次のファイルから抽出されました:

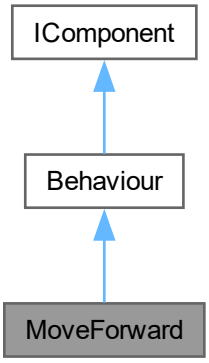
- include/components/MeshRenderer.h

6.25 MoveForward 構造体

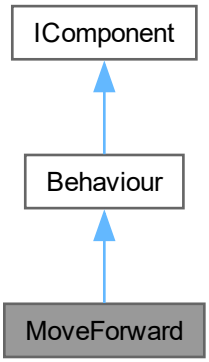
前に進むBehaviour

#include <ComponentSamples.h>

MoveForward の継承関係図



MoveForward 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speed = 2.0f
  前進速度（単位/秒）

6.25.1 詳解

前に進むBehaviour

Z 軸方向（前方）に一定速度で移動します。範囲外に出たら自動的に削除されます。

著者

山内陽

6.25.2 関数詳解

6.25.2.1 OnUpdate()

void MoveForward::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

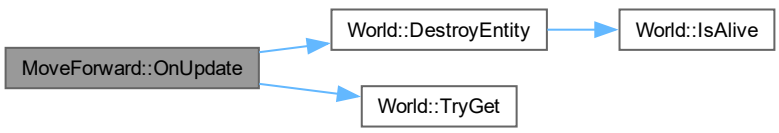
毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:



6.25.3 メンバ詳解

6.25.3.1 speed

float MoveForward::speed = 2.0f

前進速度（単位/秒）

この構造体詳解は次のファイルから抽出されました:

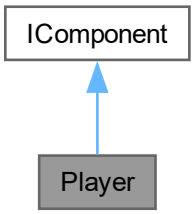
- include/samples/ComponentSamples.h

6.26 Player 構造体

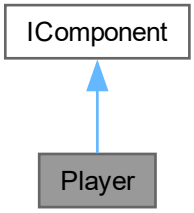
プレイヤータグ

#include <MiniGame.h>

Player の継承関係図



Player 連携図



その他の継承メンバ

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.26.1 詳解

プレイヤータグ

プレイヤーエンティティを識別するためのマーカー

この構造体詳解は次のファイルから抽出されました:

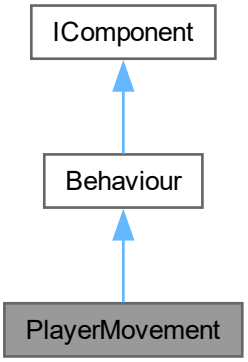
- include/scenes/MiniGame.h

6.27 PlayerMovement 構造体

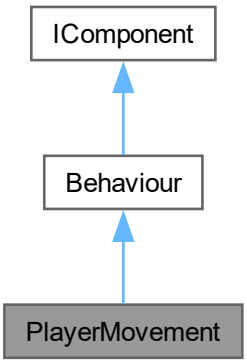
プレイヤーの移動制御Behaviour

#include <MiniGame.h>

PlayerMovement の継承関係図



PlayerMovement 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speed = 8.0f
  移動速度

6.27.1 詳解

プレイヤーの移動制御Behaviour

プレイヤーの位置を制限し、画面外に出ないようにします。

著者

山内陽

6.27.2 関数詳解

6.27.2.1 OnUpdate()

void PlayerMovement::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:

6.27.3 メンバ詳解

6.27.3.1 speed

float PlayerMovement::speed = 8.0f

移動速度

この構造体詳解は次のファイルから抽出されました:

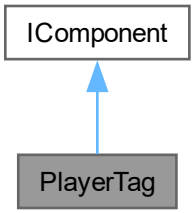
- include/scenes/MiniGame.h

6.28 PlayerTag 構造体

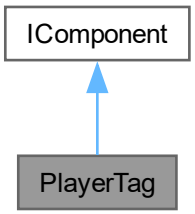
タグコンポーネント（データなし）プレイヤータグ

#include <ComponentSamples.h>

PlayerTag の継承関係図



PlayerTag 連携図



その他の継承メンバ

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

6.28.1 詳解

タグコンポーネント（データなし）プレイヤータグ

エンティティの種類を識別するためのマーカー

この構造体詳解は次のファイルから抽出されました:

- include/samples/ComponentSamples.h

6.29 RenderSystem::PSConstants 構造体

ピクセルシェーダー定数バッファ

#include <RenderSystem.h>

公開変数類

- DirectX::XMFLOAT4 color
  基本色
- float useTexture
  0= カラー, 1= テクスチャ
- float padding [3]
  パディング（16 バイトアライメント）

6.29.1 詳解

ピクセルシェーダー定数バッファ

6.29.2 メンバ詳解

6.29.2.1 color

DirectX::XMFLOAT4 RenderSystem::PSConstants::color

基本色

6.29.2.2 padding

float RenderSystem::PSConstants::padding[3]

パディング（16 バイトアライメント）

6.29.2.3 useTexture

float RenderSystem::PSConstants::useTexture

0= カラー, 1= テクスチャ

この構造体詳解は次のファイルから抽出されました:

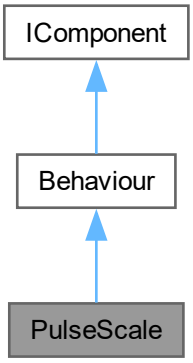
- include/graphics/RenderSystem.h

6.30 PulseScale 構造体

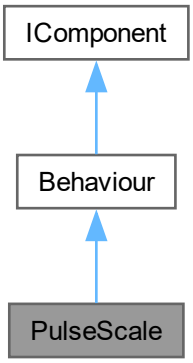
拡大縮小（パルス）Behaviour

#include <ComponentSamples.h>

PulseScale の継承関係図



PulseScale 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speed = 3.0f
  パルス速度
- float minScale = 0.5f
  最小スケール
- float maxScale = 1.5f
  最大スケール
- float time = 0.0f
  経過時間（内部管理）

6.30.1 詳解

拡大縮小（パルス）Behaviour
sin 波を使ってエンティティのスケールを周期的に変化させます。

著者

山内陽

6.30.2 関数詳解

6.30.2.1 OnUpdate()

void PulseScale::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

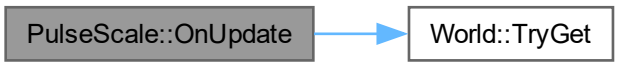
毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:



6.30.3 メンバ詳解

6.30.3.1 maxScale

float PulseScale::maxScale = 1.5f

最大スケール

6.30.3.2 minScale

float PulseScale::minScale = 0.5f

最小スケール

6.30.3.3 speed

float PulseScale::speed = 3.0f

パルス速度

6.30.3.4 time

float PulseScale::time = 0.0f

経過時間（内部管理）

この構造体詳解は次のファイルから抽出されました:

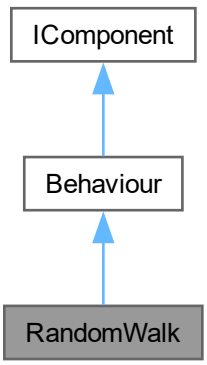
- include/samples/ComponentSamples.h

6.31 RandomWalk 構造体

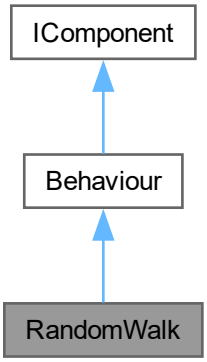
ランダムに動き回るBehaviour

#include <ComponentSamples.h>

RandomWalk の継承関係図



RandomWalk 連携図



公開メンバ関数

- void OnStart (World &w, Entity self) override
  初期化処理
- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speed = 2.0f
  移動速度
- float changeInterval = 2.0f
  方向転換の間隔（秒）
- float timer = 0.0f
  タイマー（内部管理）
- DirectX::XMFLOAT3 direction { 1.0f, 0.0f, 0.0f }
  現在の方向

6.31.1 詳解

ランダムに動き回るBehaviour

一定時間ごとにランダムな方向を選び、その方向に移動します。範囲外に出ないように制限されます。

著者

山内陽

6.31.2 関数詳解

6.31.2.1 OnStart()

void RandomWalk::OnStart (
World & w,
Entity self) [inline], [override], [virtual]

初期化処理

引数

in,out	w	
in	self	自身

Behaviourを再実装しています。

6.31.2.2 OnUpdate()

void RandomWalk::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

毎フレーム更新処理

引数

in,out	w	
in	self	自身
in	dt	デル

Behaviourを再実装しています。

呼び出し関係図:

6.31.3 メンバ詳解

6.31.3.1 changeInterval

float RandomWalk::changeInterval = 2.0f

方向転換の間隔（秒）

6.31.3.2 direction

DirectX::XMFLOAT3 RandomWalk::direction { 1.0f, 0.0f, 0.0f }

現在の方向

6.31.3.3 speed

float RandomWalk::speed = 2.0f

移動速度

6.31.3.4 timer

float RandomWalk::timer = 0.0f

タイマー（内部管理）

この構造体詳解は次のファイルから抽出されました:

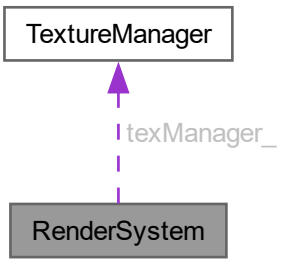
- include/samples/ComponentSamples.h

6.32 RenderSystem 構造体

テクスチャ対応レンダリングシステム

#include <RenderSystem.h>

RenderSystem 連携図



クラス

- struct PSConstants
  ピクセルシェーダー定数バッファ
- struct VSConstants
  頂点シェーダー定数バッファ

公開メンバ関数

- ~RenderSystem ()
  デストラクタ
- bool Init (GfxDevice &gfx, TextureManager &texMgr)
  初期化
- void Render (GfxDevice &gfx, World &w, const Camera &cam)
  レンダリング実行

公開変数類

- Microsoft::WRL::ComPtr< ID3D11VertexShader > vs_
  頂点シェーダー
- Microsoft::WRL::ComPtr< ID3D11PixelShader > ps_
  ピクセルシェーダー
- Microsoft::WRL::ComPtr< ID3D11InputLayout > layout_
  入力レイアウト
- Microsoft::WRL::ComPtr< ID3D11Buffer > cb_
  定数バッファ（VS 用）
- Microsoft::WRL::ComPtr< ID3D11Buffer > vb_
  頂点バッファ
- Microsoft::WRL::ComPtr< ID3D11Buffer > ib_
  インデックスバッファ
- Microsoft::WRL::ComPtr< ID3D11RasterizerState > rasterState_
  ラスタライザステート
- Microsoft::WRL::ComPtr< ID3D11SamplerState > samplerState_
  サンプラーステート
- UINT indexCount_ = 0
  インデックス数
- TextureManager * texManager_ = nullptr
  テクスチャマネージャーへのポインタ
- Microsoft::WRL::ComPtr< ID3D11Buffer > psCb_
  PS の定数バッファ

6.32.1 詳解

テクスチャ対応レンダリングシステム

ECS ワールド内のすべての描画可能なエンティティ（Transform + MeshRenderer）を自動的に描画します。単色描画とテクスチャ描画の両方に対応しています。

6.32.1.0.1 **レンダリングパイプライン**:

1. Transform から World 行列を計算
2. Camera から View・Projection 行列を取得
3. MeshRenderer の色・テクスチャ設定を適用
4. キューブメッシュを描画

使用例:

```
RenderSystem renderer;
renderer.Init(gfx, texManager);

// 毎フレーム
gfx.BeginFrame();
renderer.Render(gfx, world, camera);
gfx.EndFrame();
```

覚え書き

Transform と MeshRenderer の両方を持つエンティティのみ描画されます

著者

山内陽

6.32.2 構築子と解体子

6.32.2.1 ~RenderSystem()

RenderSystem::~RenderSystem () [inline]

デストラクタ

6.32.3 関数詳解

6.32.3.1 Init()

bool RenderSystem::Init (
GfxDevice & gfx,
TextureManager & texMgr) [inline]

初期化

引数

in	gfx	グラ
in	texMgr	テク

戻り値

bool 初期化が成功した場合は true

呼び出し関係図:

6.32.3.2 Render()

void RenderSystem::Render (
GfxDevice & gfx,
World & w,
const Camera & cam) [inline]

レンダリング実行

引数

in	gfx	グラフ
in	w	
in	cam	

ワールド内のすべてのMeshRenderer を持つエンティティを描画します。

6.32.4 メンバ詳解

6.32.4.1 cb_

Microsoft::WRL::ComPtr<ID3D11Buffer> RenderSystem::cb_

定数バッファ（VS 用）

6.32.4.2 ib_

Microsoft::WRL::ComPtr<ID3D11Buffer> RenderSystem::ib_

インデックスバッファ

6.32.4.3 indexCount_

UINT RenderSystem::indexCount_ = 0

インデックス数

6.32.4.4 layout_

Microsoft::WRL::ComPtr<ID3D11InputLayout> RenderSystem::layout_

入力レイアウト

6.32.4.5 ps_

Microsoft::WRL::ComPtr<ID3D11PixelShader> RenderSystem::ps_

ピクセルシェーダー

6.32.4.6 psCb_

Microsoft::WRL::ComPtr<ID3D11Buffer> RenderSystem::psCb_

PS の定数バッファ

6.32.4.7 rasterState_

Microsoft::WRL::ComPtr<ID3D11RasterizerState> RenderSystem::rasterState_

ラスタライザステート

6.32.4.8 samplerState_

Microsoft::WRL::ComPtr<ID3D11SamplerState> RenderSystem::samplerState_

サンプラーステート

6.32.4.9 texManager_

TextureManager* RenderSystem::texManager_ = nullptr

テクスチャマネージャーへのポインタ

6.32.4.10 vb_

Microsoft::WRL::ComPtr<ID3D11Buffer> RenderSystem::vb_

頂点バッファ

6.32.4.11 vs_

Microsoft::WRL::ComPtr<ID3D11VertexShader> RenderSystem::vs_

頂点シェーダー

この構造体詳解は次のファイルから抽出されました:

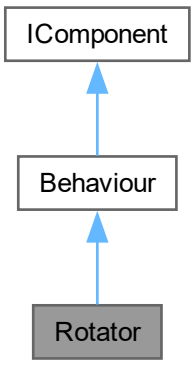
- include/graphics/RenderSystem.h

6.33 Rotator 構造体

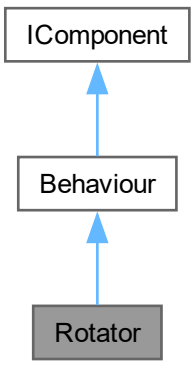
エンティティを自動的にY 軸中心で回転させるBehaviour コンポーネント

#include <Rotator.h>

Rotator の継承関係図



Rotator 連携図



公開メンバ関数

- Rotator ()=default
  デフォルトコンストラクタ
- Rotator (float s)
  回転速度を指定するコンストラクタ
- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム呼ばれる更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- float speedDegY = 45.0f
  Y 軸中心の回転速度（度/秒）

6.33.1 詳解

エンティティを自動的にY 軸中心で回転させるBehaviour コンポーネント

このコンポーネントをエンティティに追加すると、毎フレーム自動的に Y 軸（上下軸）を中心に回転します。Behaviour コンポーネントの基本的な実装例として、学習に最適です。

6.33.1.0.1 Behaviour の仕組み:

1. 毎フレーム、OnUpdate() メソッドが自動的に呼ばれる
2. OnUpdate()内で、自身のTransform コンポーネントを取得
3. Transform の rotation.y に角度を加算して回転

6.33.1.0.2 dt（デルタタイム）の重要性:

dt は前フレームからの経過秒数です。これを掛けることで、フレームレートに依存しない安定した動きを実現できます。

使用例（基本）:

```
// 毎秒 45 度で回転するキューブを作成
Entity cube = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{1, 0, 0})
    .With<Rotator>(45.0f)  // 毎秒 45 度
    .Build();
```

使用例（高速回転）:

```
// 毎秒 180 度で高速回転
Entity fastCube = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{0, 1, 0})
    .With<Rotator>(180.0f)
    .Build();
```

使用例（実行時の速度変更）:

```
// 実行中に回転速度を変更
auto* rotator = world.TryGet<Rotator>(entity);
if (rotator) {
    rotator->speedDegY = 90.0f;  // 毎秒 90 度に変更
}
```

覚え書き

Transform コンポーネントと組み合わせて使用する必要があります

参照

Behaviour Behaviour コンポーネントの基底クラス

Transform 位置・回転・スケールコンポーネント

著者

山内陽

6.33.2 構築子と解体子

6.33.2.1 Rotator() [1/2]

Rotator::Rotator () [default]

デフォルトコンストラクタ

回転速度を 45.0 度/秒に設定します

6.33.2.2 Rotator() [2/2]

Rotator::Rotator (
float s) [inline], [explicit]

回転速度を指定するコンストラクタ

引数

in	s	回転速度（

使用例:

```
Rotator slowRotator(30.0f);  // 遅い回転
Rotator fastRotator(120.0f); // 速い回転
```

6.33.3 関数詳解

6.33.3.1 OnUpdate()

void Rotator::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

毎フレーム呼ばれる更新処理

引数

in,out	w	ワー
in	self	この
in	dt	

この関数が毎フレーム自動的に呼ばれ、以下の処理を行います：

1. 自身のTransform コンポーネントを取得
2. rotation.y に speedDegY ∗ dt を加算して回転
3. 360 度を超えたら正規化（0〜360 度の範囲に収める）

覚え書き

dt を掛けることで、フレームレートに依存しない動きを実現

処理の詳細:

```
// 例: speedDegY = 45.0f, dt = 0.016 秒（60FPS）の場合
rotation.y += 45.0f * 0.016f = 0.72 度加算

// 60FPS で動作すると、1 秒間に
// 60 フレーム * 0.72 度 = 約 43.2 度回転（誤差は浮動小数点演算による）
```

Behaviourを再実装しています。

呼び出し関係図:

6.33.4 メンバ詳解

6.33.4.1 speedDegY

float Rotator::speedDegY = 45.0f

Y 軸中心の回転速度（度/秒）

毎秒何度回転するかを指定します。

- 正の値: 時計回り（右回り）
- 負の値: 反時計回り（左回り）
- 0: 回転しない

参考値:

- 45.0f: ゆっくり回転（8 秒で 1 周）
- 90.0f: 普通の速度（4 秒で 1 周）
- 180.0f: 速い回転（2 秒で 1 周）
- 360.0f: 高速回転（1 秒で 1 周）

覚え書き

デフォルトは 45.0（毎秒 45 度）

この構造体詳解は次のファイルから抽出されました:

- include/components/Rotator.h

6.34 SceneManager クラス

ゲームシーンの切り替えを管理するクラス

#include <SceneManager.h>

公開メンバ関数

- void Init (IScene *startScene, World &world)
  初期化（最初のシーンを設定）
- void RegisterScene (const char *name, IScene *scene)
  シーンを登録（名前でシーンを切り替えられるようにする）
- void Update (World &world, InputSystem &input, float deltaTime)
  毎フレームの更新
- void ChangeScene (const char *sceneName, World &world)
  シーンを切り替え
- ~SceneManager ()
  デストラクタ

6.34.1 詳解

ゲームシーンの切り替えを管理するクラス

複数のシーンを登録し、名前で切り替えを行います。

6.34.1.0.1 基本的な使い方:

1. シーンを作成
2. RegisterScene()で登録
3. Init()で開始シーンを設定
4. 毎フレームUpdate() を呼ぶ
5. 必要に応じてChangeScene() で切り替え

使用例:

```
SceneManager sceneManager;
World world;

// シーンを登録
sceneManager.RegisterScene("Title", new TitleScene());
sceneManager.RegisterScene("Game", new GameScene());
sceneManager.RegisterScene("Result", new ResultScene());

// タイトルシーンから開始
sceneManager.Init(sceneManager.GetScene("Title"), world);

// ゲームループ
while (running) {
    sceneManager.Update(world, input, deltaTime);
}
```

著者

山内陽

6.34.2 構築子と解体子

6.34.2.1 ~SceneManager()

SceneManager::~SceneManager () [inline]

デストラクタ

登録されたすべてのシーンを削除します

6.34.3 関数詳解

6.34.3.1 ChangeScene()

```
void SceneManager::ChangeScene (
            const char * sceneName,
            World & world)    [inline]
```

シーンを切り替え

引数

in	sceneName
in,out	world

現在のシーンを終了し、新しいシーンを開始します。

6.34.3.1.1 **処理の流れ**:

1. 現在のシーンのOnExit() を呼ぶ
2. 新しいシーンのOnEnter() を呼ぶ

使用例:

```
// ゲームシーンに切り替え
sceneManager.ChangeScene("Game", world);
```

6.34.3.2 Init()

```
void SceneManager::Init (
            IScene * startScene,
            World & world)    [inline]
```

初期化（最初のシーンを設定）

引数

in	startScene
in,out	world

SceneManager を初期化し、最初のシーンを開始します。startScene のOnEnter() が自動的に呼ばれます。

6.34.3.3 RegisterScene()

```
void SceneManager::RegisterScene (
            const char * name,
            IScene * scene)    [inline]
```

シーンを登録（名前でシーンを切り替えられるようにする）

引数

in	name	シ
in	scene	シーン

シーンを名前で登録します。ChangeScene()で名前を指定してシーンを切り替えられます。

使用例:

```
sceneManager.RegisterScene("Title", new TitleScene());
sceneManager.RegisterScene("Game", new GameScene());
```

6.34.3.4 Update()

void SceneManager::Update (
World & world,
InputSystem & input,
float deltaTime) [inline]

毎フレームの更新

引数

in,out	world
in	input
in	deltaTime

現在のシーンを更新し、シーン遷移をチェックします。ShouldChangeScene() が true なら自動的にシーンを切り替えます。呼び出し関係図:

このクラス詳解は次のファイルから抽出されました:

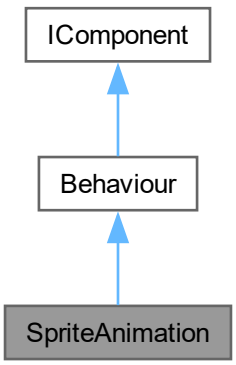
- include/scenes/SceneManager.h

6.35 SpriteAnimation 構造体

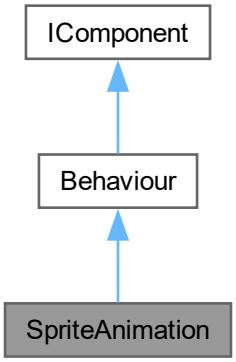
スプライトアニメーション（複数テクスチャの切り替え）コンポーネント

#include <Animation.h>

SpriteAnimation の継承関係図



SpriteAnimation 連携図



公開メンバ関数

- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理
- TextureManager::TextureHandle GetCurrentTexture () const
  現在のテクスチャを取得
- void Play ()
  アニメーションを再生
- void Stop ()
  アニメーションを停止
- void Reset ()
  アニメーションをリセット

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- std::vector< TextureManager::TextureHandle > frames
  アニメーションフレーム（テクスチャ配列）
- float frameTime = 0.1f
  1 フレームの表示時間（秒）
- bool loop = true
  ループ再生するか
- bool playing = true
  再生中か
- float currentTime = 0.0f
  内部時間（触らなくてOK）
- size_t currentFrame = 0
  現在のフレーム番号
- bool finished = false
  アニメーション終了フラグ

6.35.1 詳解

スプライトアニメーション（複数テクスチャの切り替え）コンポーネント

複数のテクスチャを順番に切り替えてパラパラアニメーションを実現します。ループ再生や 1 回だけの再生にも対応しています。

使用例（基本）:

```
SpriteAnimation anim;
anim.frames = { tex1, tex2, tex3, tex4 };  // 4 フレーム
anim.frameTime = 0.1f;                     // 1 フレーム 0.1 秒（10fps）
anim.loop = true;                          // ループ再生
world.Add<SpriteAnimation>(entity, anim);
```

使用例（毎フレーム現在のテクスチャを取得）:

```
auto* anim = world.TryGet<SpriteAnimation>(entity);
auto* renderer = world.TryGet<MeshRenderer>(entity);
if (anim && renderer) {
    renderer->texture = anim->GetCurrentTexture();
}
```

参照

UVAnimation UV スクロールアニメーション

著者

山内陽

6.35.2 関数詳解

6.35.2.1 GetCurrentTexture()

TextureManager::TextureHandle SpriteAnimation::GetCurrentTexture () const [inline]

現在のテクスチャを取得

戻り値

TextureManager::TextureHandle 現在表示すべきテクスチャ

6.35.2.2 OnUpdate()

void SpriteAnimation::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

毎フレーム更新処理

引数

in,out	w	
in	self	この
in	dt	

Behaviourを再実装しています。

6.35.2.3 Play()

void SpriteAnimation::Play () [inline]

アニメーションを再生

6.35.2.4 Reset()

void SpriteAnimation::Reset () [inline]

アニメーションをリセット

6.35.2.5 Stop()

void SpriteAnimation::Stop () [inline]

アニメーションを停止

6.35.3 メンバ詳解

6.35.3.1 currentFrame

size_t SpriteAnimation::currentFrame = 0

現在のフレーム番号

6.35.3.2 currentTime

float SpriteAnimation::currentTime = 0.0f

内部時間（触らなくてOK）

6.35.3.3 finished

bool SpriteAnimation::finished = false

アニメーション終了フラグ

6.35.3.4 frames

std::vector<TextureManager::TextureHandle> SpriteAnimation::frames

アニメーションフレーム（テクスチャ配列）

6.35.3.5 frameTime

float SpriteAnimation::frameTime = 0.1f

1 フレームの表示時間（秒）

6.35.3.6 loop

bool SpriteAnimation::loop = true

ループ再生するか

6.35.3.7 playing

bool SpriteAnimation::playing = true

再生中か

この構造体詳解は次のファイルから抽出されました:

- include/animation/Animation.h

6.36 TextureManager クラス

テクスチャ管理システム

#include <TextureManager.h>

公開型

- using TextureHandle = uint32_t
  テクスチャを識別するハンドル

公開メンバ関数

- bool Init (GfxDevice &gfx)
  初期化
- TextureHandle LoadFromFile (const char *filepath)
  ファイルからテクスチャを読み込み（BMP, PNG, JPG など）
- TextureHandle CreateTextureFromMemory (const uint8_t *data, uint32_t width, uint32_t height, uint32_t channels)
  メモリからテクスチャを作成
- ID3D11ShaderResourceView * GetSRV (TextureHandle handle) const
  テクスチャの取得
- TextureHandle GetDefaultWhite () const
  デフォルトテクスチャ（白）を取得
- void Release (TextureHandle handle)
  テクスチャの解放
- ~TextureManager ()
  デストラクタ

静的公開変数類

- static constexpr TextureHandle INVALID_TEXTURE = 0
  無効なテクスチャを表す特殊値

6.36.1 詳解

テクスチャ管理システム

テクスチャの読み込み、作成、管理を一元的に行うクラスです。ハンドルベースの管理により、安全かつ効率的にテクスチャを扱えます。

6.36.1.0.1 対応フォーマット:

- BMP (ビットマップ)
- PNG (Portable Network Graphics)
- JPG/JPEG (Joint Photographic Experts Group)
- その他 WIC がサポートする形式

6.36.1.0.2 使用例:

```
TextureManager texManager;
texManager.Init(gfx);

// ファイルから読み込み
auto handle = texManager.LoadFromFile("assets/brick.png");

// メッシュレンダラーに設定
auto* renderer = world.TryGet<MeshRenderer>(entity);
if (renderer) {
    renderer->texture = handle;
}

// テクスチャの解放
texManager.Release(handle);
```

覚え書き

すべてのテクスチャは RGBA32 フォーマットに変換されます

著者

山内陽

6.36.2 型定義メンバ詳解

6.36.2.1 TextureHandle

using TextureManager::TextureHandle = uint32_t

テクスチャを識別するハンドル

6.36.3 構築子と解体子

6.36.3.1 ~TextureManager()

TextureManager::~TextureManager () [inline]

デストラクタ

6.36.4 関数詳解

6.36.4.1 CreateTextureFromMemory()

TextureHandle TextureManager::CreateTextureFromMemory (
const uint8_t * data,
uint32_t width,
uint32_t height,
uint32_t channels) [inline]

メモリからテクスチャを作成

引数

in	data	
in	width	
in	height	
in	channels	チャ

戻り値

TextureHandle テクスチャハンドル（失敗時は INVALID_TEXTURE）

メモリ上のピクセルデータから DirectX11 テクスチャを作成します。プロシージャルテクスチャの生成などに使用できます。

6.36.4.2 GetDefaultWhite()

TextureHandle TextureManager::GetDefaultWhite () const [inline]

デフォルトテクスチャ（白）を取得

戻り値

TextureHandle 白色テクスチャのハンドル

システムが自動的に作成する 1x1 の白色テクスチャです。テクスチャが指定されていない場合のフォールバックに使用できます。

6.36.4.3 GetSRV()

ID3D11ShaderResourceView * TextureManager::GetSRV (
TextureHandle handle) const [inline]

テクスチャの取得

引数

	in	handle	テク

戻り値

ID3D11ShaderResourceView* シェーダーリソースビュー（失敗時は nullptr）

ハンドルから ShaderResourceView を取得します。これをシェーダーにバインドすることでテクスチャを使用できます。

6.36.4.4 Init()

bool TextureManager::Init (
GfxDevice & gfx) [inline]

初期化

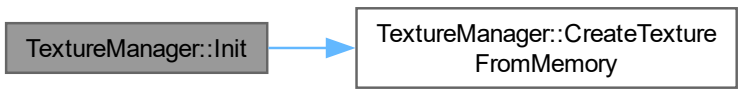
引数

	in	gfx	グラフィ

戻り値

bool 初期化が成功した場合は true

呼び出し関係図:



6.36.4.5 LoadFromFile()

TextureHandle TextureManager::LoadFromFile (
const char * filepath) [inline]

ファイルからテクスチャを読み込み（BMP, PNG, JPG など）

引数

in	filepath	画像	

戻り値

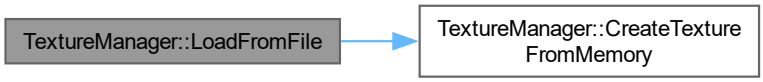
TextureHandle テクスチャハンドル（失敗時は INVALID_TEXTURE）

Windows Imaging Component (WIC) を使用して画像を読み込み、DirectX11 テクスチャに変換します。

使用例:

```
auto texture = texManager.LoadFromFile("assets/player.png");
if (texture != TextureManager::INVALID_TEXTURE) {
    // テクスチャの使用
}
```

呼び出し関係図:



6.36.4.6 Release()

void TextureManager::Release (
TextureHandle handle) [inline]

テクスチャの解放

引数

in	handle	テク	

指定されたテクスチャをメモリから解放します。解放後、そのハンドルは無効になります。

6.36.5 メンバ詳解

6.36.5.1 INVALID_TEXTURE

TextureHandle TextureManager::INVALID_TEXTURE = 0 [static], [constexpr]

無効なテクスチャを表す特殊値

このクラス詳解は次のファイルから抽出されました:

- include/graphics/TextureManager.h

6.37 Transform 構造体

3D 空間におけるエンティティの位置・回転・スケールを管理するデータコンポーネント

#include <Transform.h>

公開変数類

- DirectX::XMFLOAT3 position { 0, 0, 5 }
  エンティティの 3D 空間における位置座標
- DirectX::XMFLOAT3 rotation { 0, 0, 0 }
  エンティティの回転角度（度数法）
- DirectX::XMFLOAT3 scale { 1, 1, 1 }
  エンティティのスケール（拡大縮小率）

6.37.1 詳解

3D 空間におけるエンティティの位置・回転・スケールを管理するデータコンポーネント

このコンポーネントは 3D 空間における「場所」「向き」「大きさ」を表す基本データです。すべての 3D オブジェクトに必要となる空間情報を保持します。

6.37.1.0.1 座標系について:

- X 軸: 右方向が正（カメラから見て）
- Y 軸: 上方向が正
- Z 軸: 奥方向が正（カメラの視線方向）

6.37.1.0.2 回転の適用順序:

Y 軸回転 → X 軸回転 → Z 軸回転の順で適用されます

6.37.1.0.3 使用例:

```
// エンティティを作成し、Transform を設定
Entity cube = world.Create()
    .With<Transform>(
        DirectX::XMFLOAT3{0, 5, 0},    // 位置: Y 軸上に 5 単位
        DirectX::XMFLOAT3{0, 45, 0},   // 回転: Y 軸中心に 45 度
        DirectX::XMFLOAT3{2, 2, 2}     // スケール: 2 倍
    )
    .Build();

// 既存の Transform を取得して変更
auto* transform = world.TryGet<Transform>(cube);
if (transform) {
    transform->position.y += 1.0f;  // 上に 1 単位移動
    transform->rotation.y += 90.0f; // さらに 90 度回転
}
```

覚え書き

すべての 3D オブジェクトに推奨されるコンポーネントです

警告

回転角度は度数法（0-360 度）で指定します（ラジアンではありません）

参照

MeshRenderer 描画に使用するコンポーネント
Rotator 自動回転を行うBehaviour コンポーネント

著者

山内陽

6.37.2 メンバ詳解

6.37.2.1 position

DirectX::XMFLOAT3 Transform::position { 0, 0, 5 }

エンティティの 3D 空間における位置座標
ワールド座標系における絶対位置を表します。

- position.x: 左右位置（負の値が左、正の値が右）
- position.y: 上下位置（負の値が下、正の値が上）
- position.z: 前後位置（負の値が手前、正の値が奥）

デフォルトではカメラから 5 単位奥（Z 軸正方向）に配置されます。

覚え書き

単位系は特に定義されていませんが、通常メートル単位として扱います

6.37.2.2 rotation

DirectX::XMFLOAT3 Transform::rotation { 0, 0, 0 }

エンティティの回転角度（度数法）
オイラー角による回転を表します。各軸周りの回転角度を度数法で指定します。

- rotation.x: ピッチ（上下回転、X 軸周り）
- rotation.y: ヨー（左右回転、Y 軸周り）
- rotation.z: ロール（横転回転、Z 軸周り）

回転の適用順序: Y 軸 → X 軸 → Z 軸

覚え書き

角度は度数法（0-360 度）で指定します

警告

大きな回転を行う場合、ジンバルロックに注意してください

使用例:

```
transform.rotation = DirectX::XMFLOAT3{0, 90, 0};  // Y 軸中心に 90 度回転
```

6.37.2.3 scale

DirectX::XMFLOAT3 Transform::scale { 1, 1, 1 }

エンティティのスケール（拡大縮小率）

各軸方向の拡大縮小率を指定します。

- scale.x: X 軸方向のスケール（幅）
- scale.y: Y 軸方向のスケール（高さ）
- scale.z: Z 軸方向のスケール（奥行き）

1.0 が等倍、2.0 が 2 倍、0.5 が半分のサイズになります。負の値を指定すると反転します。

覚え書き

各軸独立してスケーリング可能です

警告

0 を指定すると描画されなくなります

使用例:

```
transform.scale = DirectX::XMFLOAT3{2, 1, 1};  // 横幅だけ 2 倍
transform.scale = DirectX::XMFLOAT3{0.5, 0.5, 0.5};  // 全体を半分のサイズに
```

この構造体詳解は次のファイルから抽出されました:

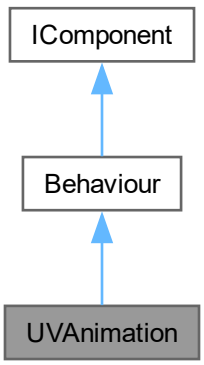
- include/components/Transform.h

6.38 UVAnimation 構造体

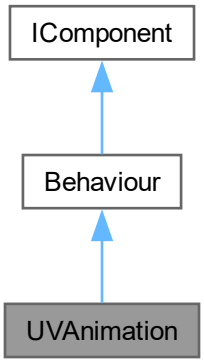
UV スクロールアニメーション（テクスチャ移動）コンポーネント

#include <Animation.h>

UVAnimation の継承関係図



UVAnimation 連携図



公開メンバ関数

- UVAnimation ()=default
  デフォルトコンストラクタ
- UVAnimation (const DirectX::XMFLOAT2 &speed)
  スクロール速度を指定するコンストラクタ
- UVAnimation (float u, float v)
  個別に U,V を指定するコンストラクタ
- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム更新処理

基底クラス Behaviour に属する継承公開メンバ関数

- virtual void OnStart (World &w, Entity self)
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- DirectX::XMFLOAT2 scrollSpeed { 0.0f, 0.0f }
  UV 座標のスクロール速度（単位/秒）
- DirectX::XMFLOAT2 currentOffset { 0.0f, 0.0f }
  現在のオフセット（自動更新）

6.38.1 詳解

UV スクロールアニメーション（テクスチャ移動）コンポーネント

テクスチャをスクロールさせて流れているように見せます。用途: 流れる水、回るタイヤ、ベルトコンベア等

使用例（横にスクロール）:

```
UVAnimation uv;
uv.scrollSpeed = DirectX::XMFLOAT2{0.5f, 0.0f};  // 毎秒 0.5 横に移動
world.Add<UVAnimation>(entity, uv);
```

使用例（簡略版）:

```
world.Add<UVAnimation>(entity, UVAnimation{0.5f, 0.0f});
```

使用例（MeshRenderer に反映）:

```
auto* uv = world.TryGet<UVAnimation>(entity);
auto* renderer = world.TryGet<MeshRenderer>(entity);
if (uv && renderer) {
    renderer->uvOffset = uv->currentOffset;
}
```

参照

SpriteAnimation スプライトアニメーション

著者

山内陽

6.38.2 構築子と解体子

6.38.2.1 UVAnimation() [1/3]

UVAnimation::UVAnimation () [default]

デフォルトコンストラクタ

6.38.2.2 UVAnimation() [2/3]

UVAnimation::UVAnimation (
const DirectX::XMFLOAT2 & speed) [inline], [explicit]

スクロール速度を指定するコンストラクタ

引数

in	speed	スクロ

6.38.2.3 UVAnimation() [3/3]

UVAnimation::UVAnimation (
float u,
float v) [inline]

個別に U,V を指定するコンストラクタ

引数

in	u	横方向スク
in	v	縦方向スク

6.38.3 関数詳解

6.38.3.1 OnUpdate()

void UVAnimation::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

毎フレーム更新処理

引数

in,out	w	
in	self	この
in	dt	

Behaviourを再実装しています。

6.38.4 メンバ詳解

6.38.4.1 currentOffset

DirectX::XMFLOAT2 UVAnimation::currentOffset { 0.0f, 0.0f }

現在のオフセット（自動更新）

6.38.4.2 scrollSpeed

DirectX::XMFLOAT2 UVAnimation::scrollSpeed { 0.0f, 0.0f }

UV 座標のスクロール速度（単位/秒）

この構造体詳解は次のファイルから抽出されました:

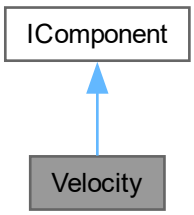
- include/animation/Animation.h

6.39 Velocity 構造体

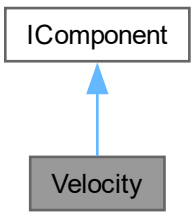
速度コンポーネント

#include <ComponentSamples.h>

Velocity の継承関係図



Velocity 連携図



公開メンバ関数

- void AddVelocity (float x, float y, float z)
  速度を加算

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- DirectX::XMFLOAT3 velocity { 0.0f, 0.0f, 0.0f }
  速度ベクトル

6.39.1 詳解

速度コンポーネント

エンティティの速度ベクトルを保持します。

著者

山内陽

6.39.2 関数詳解

6.39.2.1 AddVelocity()

```
void Velocity::AddVelocity (
                float x,
                float y,
                float z)   [inline]
```

速度を加算

引数

in	x	X 方向の速
in	y	Y 方向の速
in	z	Z 方向の速

6.39.3 メンバ詳解

6.39.3.1 velocity

```
DirectX::XMFLOAT3 Velocity::velocity { 0.0f, 0.0f, 0.0f }
```

速度ベクトル

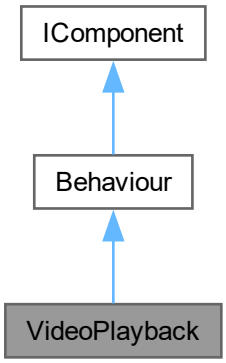
この構造体詳解は次のファイルから抽出されました:

- include/samples/ComponentSamples.h

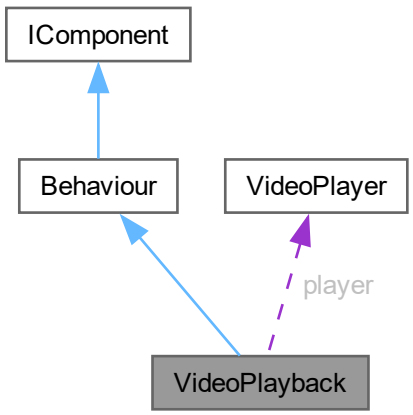
6.40 VideoPlayback 構造体

#include <VideoPlayer.h>

VideoPlayback の継承関係図



VideoPlayback 連携図



公開メンバ関数

- void OnStart (World &w, Entity self) override
  エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド
- void OnUpdate (World &w, Entity self, float dt) override
  毎フレーム呼ばれる更新メソッド

基底クラス IComponent に属する継承公開メンバ関数

- virtual ~IComponent ()=default
  仮想デストラクタ

公開変数類

- VideoPlayer * player = nullptr
- bool autoPlay = true

6.40.1 関数詳解

6.40.1.1 OnStart()

void VideoPlayback::OnStart (
World & w,
Entity self) [inline], [override], [virtual]

エンティティ作成直後に 1 度だけ呼ばれる初期化メソッド

引数

in	w	ワールド
in	self	

初期化処理が必要な場合にオーバーライドします。例: 初期位置の設定、初期状態の計算など

覚え書き

デフォルト実装は何もしません

使用例:

```
struct MyBehaviour : Behaviour {
    void OnStart(World& w, Entity self) override {
        // 初期化処理
        auto* transform = w.TryGet<Transform>(self);
        if (transform) {
            transform->position.y = 5.0f;  // 初期位置を設定
        }
    }
};
```

Behaviourを再実装しています。

6.40.1.2 OnUpdate()

void VideoPlayback::OnUpdate (
World & w,
Entity self,
float dt) [inline], [override], [virtual]

毎フレーム呼ばれる更新メソッド

引数

in,out	w	ワー
in	self	
in	dt	

ゲームロジックを実装する際にオーバーライドします。dt を使うことで、フレームレートに依存しない処理を実現できます。

覚え書き

デフォルト実装は何もしません

使用例:

```
struct MoveForward : Behaviour {
    float speed = 5.0f;

    void OnUpdate(World& w, Entity self, float dt) override {
        auto* transform = w.TryGet<Transform>(self);
        if (transform) {
            // dt を使ってフレームレート非依存な移動
            transform->position.z += speed * dt;
        }
    }
};
```

Behaviourを再実装しています。

6.40.2 メンバ詳解

6.40.2.1 autoPlay

bool VideoPlayback::autoPlay = true

6.40.2.2 player

VideoPlayer* VideoPlayback::player = nullptr

この構造体詳解は次のファイルから抽出されました:

- include/graphics/VideoPlayer.h

6.41 VideoPlayer クラス

#include <VideoPlayer.h>

公開メンバ関数

- bool Init ()
- ~VideoPlayer ()
- bool Open (GfxDevice &gfx, const char *filepath)
- bool Update (float dt)
- void Play ()
- void Stop ()
- void SetLoop (bool loop)
- ID3D11ShaderResourceView * GetSRV () const
- bool IsPlaying () const
- UINT GetWidth () const
- UINT GetHeight () const

6.41.1 構築子と解体子

6.41.1.1 ~VideoPlayer()

VideoPlayer::~VideoPlayer () [inline]

6.41.2 関数詳解

6.41.2.1 GetHeight()

UINT VideoPlayer::GetHeight () const [inline]

6.41.2.2 GetSRV()

ID3D11ShaderResourceView * VideoPlayer::GetSRV () const [inline]

6.41.2.3 GetWidth()

UINT VideoPlayer::GetWidth () const [inline]

6.41.2.4 Init()

bool VideoPlayer::Init () [inline]

6.41.2.5 IsPlaying()

bool VideoPlayer::IsPlaying () const [inline]

6.41.2.6 Open()

bool VideoPlayer::Open (
GfxDevice & gfx,
const char * filepath) [inline]

6.41.2.7 Play()

void VideoPlayer::Play () [inline]

6.41.2.8 SetLoop()

void VideoPlayer::SetLoop (
bool loop) [inline]

6.41.2.9 Stop()

void VideoPlayer::Stop () [inline]

6.41.2.10 Update()

bool VideoPlayer::Update (
float dt) [inline]

このクラス詳解は次のファイルから抽出されました:

- include/graphics/VideoPlayer.h

6.42 RenderSystem::VSConstants 構造体

頂点シェーダー定数バッファ

#include <RenderSystem.h>

公開変数類

- DirectX::XMMATRIX WVP
  World * View * Projection 行列
- DirectX::XMFLOAT4 uvTransform
  xy=offset, zw=scale

6.42.1 詳解

頂点シェーダー定数バッファ

6.42.2 メンバ詳解

6.42.2.1 uvTransform

DirectX::XMFLOAT4 RenderSystem::VSConstants::uvTransform

xy=offset, zw=scale

6.42.2.2 WVP

DirectX::XMMATRIX RenderSystem::VSConstants::WVP

World * View * Projection 行列

この構造体詳解は次のファイルから抽出されました:

- include/graphics/RenderSystem.h

6.43 World クラス

ECS ワールド管理クラス - エンティティとコンポーネントのすべてを管理

#include <World.h>

公開メンバ関数

- Entity CreateEntity ()
  エンティティを作成する（基本版）
- EntityBuilder Create ()
  エンティティを作成する（ビルダー版）- おすすめ！
- bool IsAlive (Entity e) const
  エンティティが生存しているか確認
- void DestroyEntity (Entity e)
  エンティティを削除する
- template<class T, class... Args>
  T & Add (Entity e, Args &&...args)
  コンポーネントを追加する
- template<class T>
  T * TryGet (Entity e)
  コンポーネントを取得する（nullptr の可能性あり）
- template<class T>
  bool Remove (Entity e)
  コンポーネントを削除する
- template<class T, class F>
  void ForEach (F &&fn)
  全コンポーネントに対して処理を実行
- void Tick (float dt)
  全Behaviour の更新（毎フレーム呼ぶ）

フレンド

- class EntityBuilder
  EntityBuilder から private メンバにアクセス可能

6.43.1 詳解

ECS ワールド管理クラス - エンティティとコンポーネントのすべてを管理

World クラスは、ゲーム世界の「管理者」です。以下の機能を提供します：

- エンティティの作成・削除
- コンポーネントの追加・削除・取得
- 全Behaviour コンポーネントの更新

6.43.1.0.1 初学者向けガイド:

World は「ゲーム世界そのもの」と考えてください。

- CreateEntity() でゲームオブジェクトを作る
- Add< コンポーネント >() で機能を追加
- TryGet< コンポーネント >() で機能を取得
- Tick() で全オブジェクトを更新

基本的な使い方:

```
World world;

// エンティティ作成
Entity player = world.CreateEntity();

// コンポーネント追加
world.Add<Transform>(player, Transform{...});
world.Add<MeshRenderer>(player, MeshRenderer{...});

// コンポーネント取得
auto* transform = world.TryGet<Transform>(player);
if (transform) {
    transform->position.x += 1.0f;
}

// 毎フレーム更新
world.Tick(deltaTime);
```

ビルダーパターン（推奨）:

```
Entity player = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{0, 1, 0})
    .With<Rotator>(45.0f)
    .Build();
```

参照

Entity エンティティ構造体
IComponent コンポーネント基底クラス
Behaviour 動的コンポーネント基底クラス

著者

山内陽

6.43.2 関数詳解

6.43.2.1 Add()

```
template<class T, class... Args>
T & World::Add (
                Entity e,
                Args &&... args)    [inline]
```

コンポーネントを追加する

テンプレート引数

	T	追加するコンポーネントの型
	Args	コンストラクタ引数の型（可変長）

引数

	in	e	
	in	args	コンポ

戻り値

T& 追加されたコンポーネントへの参照

指定したエンティティにコンポーネントを追加します。Behaviour コンポーネントの場合、自動的にTick() で更新されます。

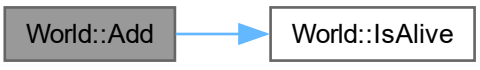
覚え書き

エンティティは生存している必要があります

使用例:

```
Entity e = world.CreateEntity();
world.Add<Transform>(e, Transform{...});
world.Add<MeshRenderer>(e, MeshRenderer{DirectX::XMFLOAT3{1, 0, 0}});
world.Add<Rotator>(e, Rotator{45.0f});
```

呼び出し関係図:



6.43.2.2 Create()

EntityBuilder World::Create () [inline]

エンティティを作成する（ビルダー版）- おすすめ！

戻り値

EntityBuilder ビルダーオブジェクト

メソッドチェーンでコンポーネントを追加できる便利な方法です。初学者にも読みやすく、おすすめの方法です。

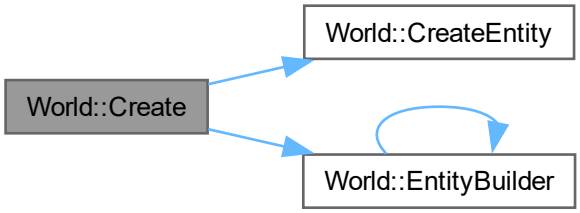
使用例:

```
Entity player = world.Create()
    .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
    .With<MeshRenderer>(DirectX::XMFLOAT3{1, 0, 0})
    .With<Rotator>(45.0f)
    .Build();
```

参照

EntityBuilder ビルダークラス

呼び出し関係図:



6.43.2.3 CreateEntity()

Entity World::CreateEntity () [inline]

エンティティを作成する（基本版）

戻り値

Entity 新しく作成されたエンティティ

一意なID を持つ新しいエンティティを作成します。この段階ではコンポーネントは何も付いていません。

覚え書き

通常はCreate()（ビルダー版）を使う方が便利です

使用例:

```
Entity e = world.CreateEntity();
world.Add<Transform>(e, Transform{...});
world.Add<MeshRenderer>(e, MeshRenderer{...});
```

6.43.2.4 DestroyEntity()

void World::DestroyEntity (
Entity e) [inline]

エンティティを削除する

引数

	in	e	削除するエ

エンティティとそれに付いているすべてのコンポーネントを削除します。削除後、そのエンティティを使用してはいけません。

警告

削除されたエンティティを使用するとクラッシュする可能性があります

使用例:

```
// 敵が倒されたら削除
auto* health = world.TryGet<Health>(enemy);
if (health && health->hp <= 0) {
    world.DestroyEntity(enemy);
}
```

呼び出し関係図:

6.43.2.5 ForEach()

template<class T, class F>
void World::ForEach (
F && fn) [inline]

全コンポーネントに対して処理を実行

テンプレート引数

	T	対象コンポーネントの型
	F	ラムダ関数の型

引数

	in	fn	実行する関

指定した型のコンポーネントを持つ全エンティティに対して処理を実行します。

使用例:

```
// 全オブジェクトを少しずつ上に移動
world.ForEach<Transform>([](Entity e, Transform& t) {
    t.position.y += 0.01f;
});

// 全敵の体力を減らす
world.ForEach<Health>([](Entity e, Health& h) {
    h.hp -= 10.0f;
});
```

呼び出し関係図:

6.43.2.6 IsAlive()

bool World::IsAlive (
Entity e) const [inline]

エンティティが生存しているか確認

引数

in	e	確認するエ

戻り値

true 生存している, false 削除済み

エンティティがDestroyEntity() で削除されていないかチェックします。

使用例:

```
if (world.IsAlive(entity)) {
    // エンティティが有効な場合の処理
}
```

6.43.2.7 Remove()

template<class T>
bool World::Remove (
Entity e) [inline]

コンポーネントを削除する

テンプレート引数

	T	削除するコンポーネントの型

引数

in	e	対象エンテ

戻り値

true 削除成功, false コンポーネントが存在しなかった

使用例:

```
world.Remove<Rotator>(entity);  // 回転を止める
```

6.43.2.8 Tick()

```
void World::Tick (
        float dt)   [inline]
```

全Behaviour の更新（毎フレーム呼ぶ）

引数

in	dt	デルタタ

すべてのBehaviour コンポーネントのOnUpdate() を呼び出します。ゲームループ内で毎フレーム呼び出してください。

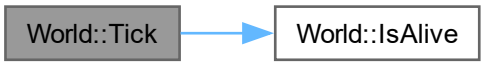
使用例:

```
// ゲームループ
while (running) {
    float deltaTime = CalculateDeltaTime();
    world.Tick(deltaTime);  // 全 Behaviour を更新
    Render();
}
```

覚え書き

初回はOnStart() も呼ばれます

呼び出し関係図:



6.43.2.9 TryGet()

```
template<class T>
T * World::TryGet (
        Entity e)   [inline]
```

コンポーネントを取得する（nullptr の可能性あり）

テンプレート引数

	T	取得するコンポーネントの型

引数

in	e	対象エンテ

戻り値

T* コンポーネントへのポインタ（存在しない場合は nullptr）

指定したエンティティからコンポーネントを取得します。コンポーネントが存在しない場合は nullptr を返すため、必ずチェックが必要です。

警告

必ず nullptr チェックを行ってください

使用例:

```
auto* transform = world.TryGet<Transform>(entity);
if (transform) {
    transform->position.x += 1.0f;  // 安全に使用
}
```

6.43.3 フレンドと関連関数の詳解

6.43.3.1 EntityBuilder

friend class EntityBuilder [friend]

EntityBuilder から private メンバにアクセス可能

このクラス詳解は次のファイルから抽出されました:

- include/ecs/World.h

Chapter 7

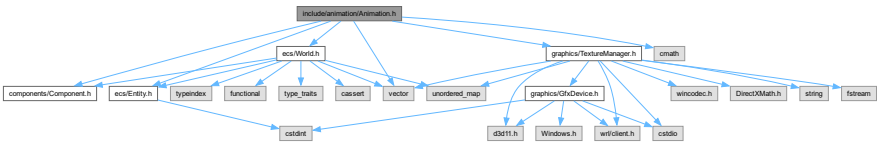
ファイル詳解

7.1 include/animation/Animation.h ファイル

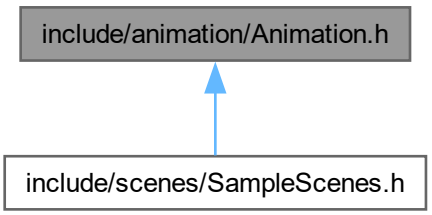
アニメーションコンポーネントの定義

```
#include ”components/Component.h”
#include ”ecs/Entity.h”
#include ”ecs/World.h”
#include ”graphics/TextureManager.h”
#include <vector>
#include <cmath>
```

Animation.h の依存先関係図:



被依存関係図:



クラス

- struct SpriteAnimation
  スプライトアニメーション（複数テクスチャの切り替え）コンポーネント
- struct UVAnimation
  UV スクロールアニメーション（テクスチャ移動）コンポーネント

7.1.1 詳解

アニメーションコンポーネントの定義

著者

山内陽

日付

2024

バージョン

5.0

このファイルはスプライトアニメーションとUV スクロールアニメーションを実現するBehaviour コンポーネントを定義します。

7.2 Animation.h

[詳解]

```
00001 #pragma once
00002 #include "components/Component.h"
00003 #include "ecs/Entity.h"
00004 #include "ecs/World.h"
00005 #include "graphics/TextureManager.h"
00006 #include <vector>
00007 #include <cmath>
00008
00020
00050 struct SpriteAnimation : Behaviour {
00051     std::vector<TextureManager::TextureHandle> frames;
00052     float frameTime = 0.1f;
00053     bool loop = true;
00054     bool playing = true;
00055
00056     float currentTime = 0.0f;
00057     size_t currentFrame = 0;
00058     bool finished = false;
00059
00066     void OnUpdate(World& w, Entity self, float dt) override {
00067         if (!playing || frames.empty()) return;
00068
00069         currentTime += dt;
00070
00071         // フレーム切り替え
00072         if (currentTime >= frameTime) {
00073             currentTime -= frameTime;
00074             currentFrame++;
00075
00076             if (currentFrame >= frames.size()) {
00077                 if (loop) {
00078                     currentFrame = 0;
00079                 } else {
```

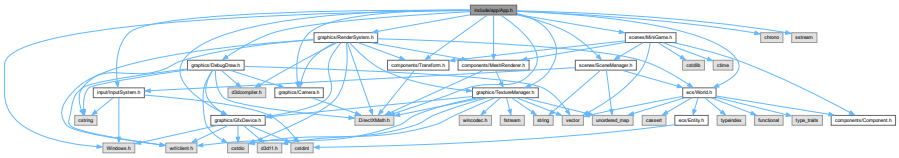
```
00080                 currentFrame = frames.size() - 1;
00081                 playing = false;
00082                 finished = true;
00083             }
00084         }
00085     }
00086 }
00087
00092 TextureManager::TextureHandle GetCurrentTexture() const {
00093     if (frames.empty()) return TextureManager::INVALID_TEXTURE;
00094     return frames[currentFrame];
00095 }
00096
00100 void Play() {
00101     playing = true;
00102     finished = false;
00103 }
00104
00108 void Stop() {
00109     playing = false;
00110 }
00111
00115 void Reset() {
00116     currentFrame = 0;
00117     currentTime = 0.0f;
00118     finished = false;
00119 }
00120 };
00121
00154 struct UVAnimation : Behaviour {
00155     DirectX::XMFLOAT2 scrollSpeed{ 0.0f, 0.0f };
00156     DirectX::XMFLOAT2 currentOffset{ 0.0f, 0.0f };
00157
00161     UVAnimation() = default;
00162
00167     explicit UVAnimation(const DirectX::XMFLOAT2& speed) : scrollSpeed(speed) {}
00168
00174     UVAnimation(float u, float v) : scrollSpeed{u, v} {}
00175
00182     void OnUpdate(World& w, Entity self, float dt) override {
00183         // スクロール量を加算
00184         currentOffset.x += scrollSpeed.x * dt;
00185         currentOffset.y += scrollSpeed.y * dt;
00186
00187         // 0-1 範囲に正規化（繰り返し）
00188         currentOffset.x = fmodf(currentOffset.x, 1.0f);
00189         currentOffset.y = fmodf(currentOffset.y, 1.0f);
00190
00191         if (currentOffset.x < 0.0f) currentOffset.x += 1.0f;
00192         if (currentOffset.y < 0.0f) currentOffset.y += 1.0f;
00193     }
00194 };
```

7.3 include/app/App.h ファイル

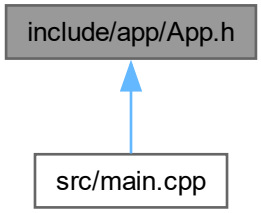
ミニゲームのメインアプリケーションクラス

```
#include <Windows.h>
#include <DirectXMath.h>
#include <chrono>
#include <sstream>
#include "graphics/GfxDevice.h"
#include "graphics/RenderSystem.h"
#include "ecs/World.h"
#include "graphics/Camera.h"
#include "input/InputSystem.h"
#include "graphics/TextureManager.h"
#include "graphics/DebugDraw.h"
#include "components/Transform.h"
#include "components/MeshRenderer.h"
#include "scenes/SceneManager.h"
```

#include ”scenes/MiniGame.h”
App.h の依存先関係図:



被依存関係図:



クラス

- struct App
  ミニゲームのメインアプリケーションクラス

マクロ定義

- #define WIN32_LEAN_AND_MEAN
- #define NOMINMAX

7.3.1 詳解

ミニゲームのメインアプリケーションクラス

著者

山内陽

日付

2024

バージョン

5.1

7.3.2 マクロ定義詳解

7.3.2.1 NOMINMAX

#define NOMINMAX

7.3.2.2 WIN32_LEAN_AND_MEAN

#define WIN32_LEAN_AND_MEAN

7.4 App.h

[詳解]

```
00001
00008 #pragma once
00009 // ============================================================
00010 // App.h - ミニゲームのアプリケーション
00011 // ============================================================
00012 // 【ゲーム内容】シンプルなシューティングゲーム
00013 // 【操作方法】A/D: 移動, スペース: 弾発射, ESC: 終了
00014 // ============================================================
00015
00016 #define WIN32_LEAN_AND_MEAN
00017 #define NOMINMAX
00018
00019 #include <Windows.h>
00020 #include <DirectXMath.h>
00021 #include <chrono>
00022 #include <sstream>
00023
00024 // DirectX11 & ECS システム
00025 #include "graphics/GfxDevice.h"
00026 #include "graphics/RenderSystem.h"
00027 #include "ecs/World.h"
00028 #include "graphics/Camera.h"
00029 #include "input/InputSystem.h"
00030 #include "graphics/TextureManager.h"
00031 #include "graphics/DebugDraw.h"
00032
00033 // コンポーネント
00034 #include "components/Transform.h"
00035 #include "components/MeshRenderer.h"
00036
00037 // ゲームシステム
00038 #include "scenes/SceneManager.h"
00039 #include "scenes/MiniGame.h"
00040
00048 struct App {
00049     // Windows 関連
00050     HWND hwnd_ = nullptr;
00051
00052     // DirectX11 システム
00053     GfxDevice gfx_;
00054     RenderSystem renderer_;
00055     TextureManager texManager_;
00056
00057     // ECS システム
00058     World world_;
00059     Camera camera_;
00060     InputSystem input_;
00061
00062     // シーン管理
00063     SceneManager sceneManager_;
00064     GameScene* gameScene_ = nullptr;
00065
00066 #ifdef _DEBUG
00067     DebugDraw debugDraw_;
00068 #endif
00069
00070     // ============================================================
00071     // 初期化
00072     // ============================================================
00080     bool Init(HINSTANCE hInst, int width = 1280, int height = 720) {
```

```
00081         CoInitializeEx(nullptr, COINIT_MULTITHREADED);
00082
00083         if (!CreateAppWindow(hInst, width, height)) {
00084             return false;
00085         }
00086
00087         if (!InitializeGraphics(width, height)) {
00088             return false;
00089         }
00090
00091         SetupCamera(width, height);
00092
00093         // ゲームシーンの初期化
00094         InitializeGame();
00095
00096         return true;
00097     }
00098
00099     // ============================================================
00100     // メインループ
00101     // ============================================================
00107     void Run() {
00108         MSG msg{};
00109         auto previousTime = std::chrono::high_resolution_clock::now();
00110
00111         while (msg.message != WM_QUIT) {
00112             // Windows メッセージ処理
00113             if (ProcessWindowsMessages(msg)) {
00114                 continue;
00115             }
00116
00117             // 時間の計算
00118             float deltaTime = CalculateDeltaTime(previousTime);
00119
00120             // 入力の更新
00121             input_.Update();
00122
00123             // ESC キーで終了
00124             if (input_.GetKeyDown(VK_ESCAPE)) {
00125                 PostQuitMessage(0);
00126             }
00127
00128             // シーンの更新
00129             sceneManager_.Update(world_, input_, deltaTime);
00130
00131             // 画面の描画
00132             RenderFrame();
00133
00134             // ウィンドウタイトルにスコア表示
00135             UpdateWindowTitle();
00136         }
00137     }
00138
00142     ~App() {
00143         CoUninitialize();
00144     }
00145
00146 private:
00147     // ============================================================
00148     // 初期化ヘルパー
00149     // ============================================================
00150
00158     bool CreateAppWindow(HINSTANCE hInst, int width, int height) {
00159         WNDCLASSEX wc{ sizeof(WNDCLASSEX) };
00160         wc.style = CS_HREDRAW | CS_VREDRAW;
00161         wc.lpfnWndProc = WndProcStatic;
00162         wc.hInstance = hInst;
00163         wc.hCursor = LoadCursor(nullptr, IDC_ARROW);
00164         wc.lpszClassName = L"MiniGame_Class";
00165
00166         if (!RegisterClassEx(&wc)) {
00167             return false;
00168         }
00169
00170         RECT rc{ 0, 0, width, height };
00171         AdjustWindowRect(&rc, WS_OVERLAPPEDWINDOW, FALSE);
00172
00173         hwnd_ = CreateWindowW(
00174             wc.lpszClassName,
00175             L" シューティングゲーム - A/D: 移動 スペース: 発射 ESC: 終了",
00176             WS_OVERLAPPEDWINDOW,
00177             CW_USEDEFAULT, CW_USEDEFAULT,
00178             rc.right - rc.left, rc.bottom - rc.top,
00179             nullptr, nullptr, hInst, this
00180         );
00181
00182         if (!hwnd_) {
```

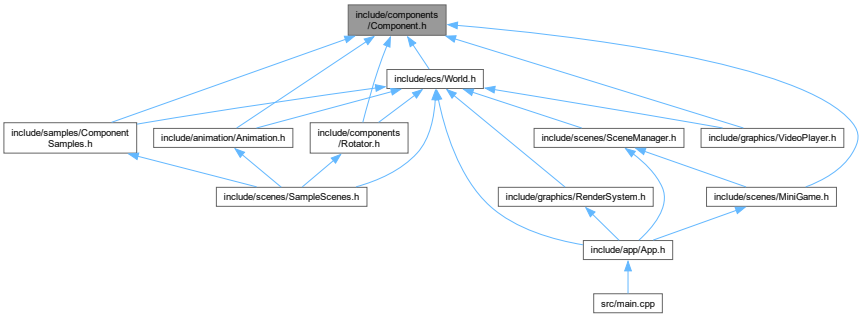
```
00183             return false;
00184         }
00185
00186         ShowWindow(hwnd_, SW_SHOW);
00187         return true;
00188     }
00189
00196     bool InitializeGraphics(int width, int height) {
00197         if (!gfx_.Init(hwnd_, width, height)) {
00198             MessageBoxA(nullptr, "DirectX11 の初期化に失敗", " エラー", MB_OK | MB_ICONERROR);
00199             return false;
00200         }
00201
00202         if (!texManager_.Init(gfx_)) {
00203             MessageBoxA(nullptr, "TextureManager の初期化に失敗", " エラー", MB_OK | MB_ICONERROR);
00204             return false;
00205         }
00206
00207         if (!renderer_.Init(gfx_, texManager_)) {
00208             MessageBoxA(nullptr, "RenderSystem の初期化に失敗", " エラー", MB_OK | MB_ICONERROR);
00209             return false;
00210         }
00211
00212         input_.Init();
00213
00214 #ifdef _DEBUG
00215         if (!debugDraw_.Init(gfx_)) {
00216             MessageBoxA(nullptr, "DebugDraw の初期化に失敗", " 警告", MB_OK | MB_ICONWARNING);
00217         }
00218 #endif
00219
00220         return true;
00221     }
00222
00228     void SetupCamera(int width, int height) {
00229         float aspectRatio = static_cast<float>(width) / static_cast<float>(height);
00230
00231         camera_ = Camera::LookAtLH(
00232             DirectX::XM_PIDIV4,
00233             aspectRatio,
00234             0.1f,
00235             100.0f,
00236             DirectX::XMFLOAT3{ 0, 0, -20 },  // カメラを引いて全体が見えるように
00237             DirectX::XMFLOAT3{ 0, 0, 0 },
00238             DirectX::XMFLOAT3{ 0, 1, 0 }
00239         );
00240     }
00241
00245     void InitializeGame() {
00246         // ゲームシーンを作成
00247         gameScene_ = new GameScene();
00248
00249         // シーンマネージャーに登録
00250         sceneManager_.RegisterScene("Game", gameScene_);
00251         sceneManager_.Init(gameScene_, world_);
00252     }
00253
00254     // ============================================================
00255     // メインループのヘルパー
00256     // ============================================================
00257
00263     bool ProcessWindowsMessages(MSG& msg) {
00264         if (PeekMessage(&msg, nullptr, 0U, 0U, PM_REMOVE)) {
00265             TranslateMessage(&msg);
00266             DispatchMessage(&msg);
00267             return true;
00268         }
00269         return false;
00270     }
00271
00277     float CalculateDeltaTime(std::chrono::high_resolution_clock::time_point& previousTime) {
00278         auto currentTime = std::chrono::high_resolution_clock::now();
00279         std::chrono::duration<float> deltaTime = currentTime - previousTime;
00280         previousTime = currentTime;
00281         return deltaTime.count();
00282     }
00283
00287     void RenderFrame() {
00288         gfx_.BeginFrame();
00289
00290 #ifdef _DEBUG
00291         DrawDebugInfo();
00292 #endif
00293
00294         renderer_.Render(gfx_, world_, camera_);
00295
00296 #ifdef _DEBUG
```

```
00297         debugDraw_.Render(gfx_, camera_);
00298 #endif
00299
00300         gfx_.EndFrame();
00301     }
00302
00306     void UpdateWindowTitle() {
00307         if (gameScene_) {
00308             std::wstringstream ss;
00309             ss « L" シューティングゲーム - スコア: " « gameScene_->GetScore()
00310                « L" | A/D: 移動 スペース: 発射 ESC: 終了";
00311             SetWindowTextW(hwnd_, ss.str().c_str());
00312         }
00313     }
00314
00315 #ifdef _DEBUG
00319     void DrawDebugInfo() {
00320         debugDraw_.Clear();
00321         debugDraw_.DrawGrid(20.0f, 20, DirectX::XMFLOAT3{0.2f, 0.2f, 0.2f});
00322         debugDraw_.DrawAxes(5.0f);
00323     }
00324 #endif
00325
00326     // ========================================================
00327     // Windows メッセージ処理
00328     // ========================================================
00329
00336     static LRESULT CALLBACK WndProcStatic(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {
00337         App* app = nullptr;
00338
00339         if (msg == WM_NCCREATE) {
00340             CREATESTRUCT* cs = reinterpret_cast<CREATESTRUCT*>(lp);
00341             app = reinterpret_cast<App*>(cs->lpCreateParams);
00342             SetWindowLongPtr(hWnd, GWLP_USERDATA, reinterpret_cast<LONG_PTR>(app));
00343         } else {
00344             app = reinterpret_cast<App*>(GetWindowLongPtr(hWnd, GWLP_USERDATA));
00345         }
00346
00347         if (app) {
00348             return app->WndProc(hWnd, msg, wp, lp);
00349         }
00350
00351         return DefWindowProc(hWnd, msg, wp, lp);
00352     }
00353
00358     LRESULT WndProc(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {
00359         switch (msg) {
00360         case WM_DESTROY:
00361             PostQuitMessage(0);
00362             return 0;
00363
00364         case WM_MOUSEWHEEL:
00365             input_.OnMouseWheel(GET_WHEEL_DELTA_WPARAM(wp));
00366             return 0;
00367         }
00368
00369         return DefWindowProc(hWnd, msg, wp, lp);
00370     }
00371 };
00372
00373 // ========================================================
00374 // 作成者: 山内陽
00375 // バージョン: v5.1 - Doxygen コメントを追加
00376 // ========================================================
```

7.5 include/components/Component.h ファイル

ECS コンポーネントシステムの基底クラスとマクロ定義

被依存関係図:



クラス

- interface IComponent
  前方宣言: Entity 構造体
- class Behaviour
  毎フレーム更新される動的コンポーネントの基底クラス

マクロ定義

- #define DEFINE_DATA_COMPONENT(ComponentName, ...)
  データコンポーネントを簡単に定義するマクロ
- #define DEFINE_BEHAVIOUR(BehaviourName, DataMembers, UpdateCode)
  Behaviour コンポーネントを簡単に定義するマクロ

7.5.1 詳解

ECS コンポーネントシステムの基底クラスとマクロ定義

著者

山内陽

日付

2024

バージョン

5.0

このファイルは、ECS アーキテクチャにおけるコンポーネントの基底クラスと、コンポーネントを簡単に定義するためのマクロを提供します。

7.5.1.0.1 コンポーネントとは:

ゲームオブジェクトに付ける「部品」のこと。例: 「位置」「見た目」「動き」などを別々のコンポーネントとして管理

7.5.1.0.2 2 種類のコンポーネント:

1. **データコンポーネント**: データのみを保持（例: Transform, Health）
2. Behaviour **コンポーネント**: 毎フレーム実行される処理を持つ（例: Rotator, PlayerMovement）

7.5.2 マクロ定義詳解

7.5.2.1 DEFINE_BEHAVIOUR

```
#define DEFINE_BEHAVIOUR(
                BehaviourName,
                DataMembers,
                UpdateCode)
```

値:

```
struct BehaviourName : Behaviour { \
    DataMembers \
    void OnUpdate(World& w, Entity self, float dt) override { \
        UpdateCode \
    } \
}
```

Behaviour コンポーネントを簡単に定義するマクロ

引数

BehaviourName	コンポーネントの名前
DataMembers	メンバ変数の定義
UpdateCode	OnUpdate() 内で実行するコード

動的なコンポーネントを簡潔に定義できます。OnUpdate() の実装を直接書けるため、コードが読みやすくなります。

警告

DataMembers とUpdateCode の間にはカンマが必要です

使用例:

```
// 上下に揺れるコンポーネント
DEFINE_BEHAVIOUR(Bouncer,
    float speed = 1.0f;
    float time = 0.0f;
,
    time += dt * speed;
    auto* t = w.TryGet<Transform>(self);
    if (t) t->position.y = sinf(time);
)

// 前進するコンポーネント
DEFINE_BEHAVIOUR(MoveForward,
    float speed = 5.0f;
,
    auto* t = w.TryGet<Transform>(self);
    if (t) t->position.z += speed * dt;
)
```

覚え書き

OnStart() は定義できません。必要な場合は通常の継承を使用してください。

著者

山内陽

7.5.2.2 DEFINE_DATA_COMPONENT

```
#define DEFINE_DATA_COMPONENT(
                ComponentName,
                ...)
```

値:

```
struct ComponentName : IComponent { \
    ___VA_ARGS___ \
}
```

データコンポーネントを簡単に定義するマクロ

引数

ComponentName	コンポーネントの名前
...	メンバ変数の定義（複数可）

データのみを持つコンポーネントを 1 行で定義できます。ボイラープレートコードを書かずに済むため、初学者でも簡単に使えます。

使用例:

```
// 体力コンポーネント
DEFINE_DATA_COMPONENT(Health,
    float hp = 100.0f;
    float maxHp = 100.0f;
)

// 速度コンポーネント
DEFINE_DATA_COMPONENT(Velocity,
    DirectX::XMFLOAT3 velocity{0, 0, 0};
)

// タグコンポーネント（データなし）
DEFINE_DATA_COMPONENT(Player, )
```

覚え書き

セミコロンは不要です（マクロ内で自動的に追加されます）

著者

山内陽

7.6 Component.h

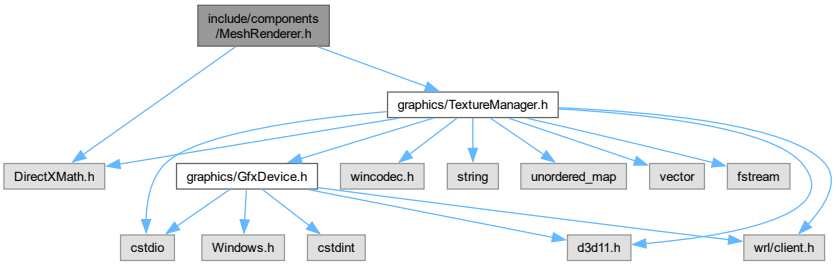
[詳解]

```
00001 #pragma once
00002
00022
00023 class World;
00024 struct Entity;
00025
00048 struct IComponent {
00053     virtual ~IComponent() = default;
00054 };
00055
00094 struct Behaviour : IComponent {
00120     virtual void OnStart(World& w, Entity self) {}
00121
00150     virtual void OnUpdate(World& w, Entity self, float dt) {}
00151 };
00152
00185 #define DEFINE_DATA_COMPONENT(ComponentName, ...) \
00186     struct ComponentName : IComponent { \
00187         __VA_ARGS__ \
00188     }
00189
00229 #define DEFINE_BEHAVIOUR(BehaviourName, DataMembers, UpdateCode) \
00230     struct BehaviourName : Behaviour { \
00231         DataMembers \
00232         void OnUpdate(World& w, Entity self, float dt) override { \
00233             UpdateCode \
00234         } \
00235     }
```

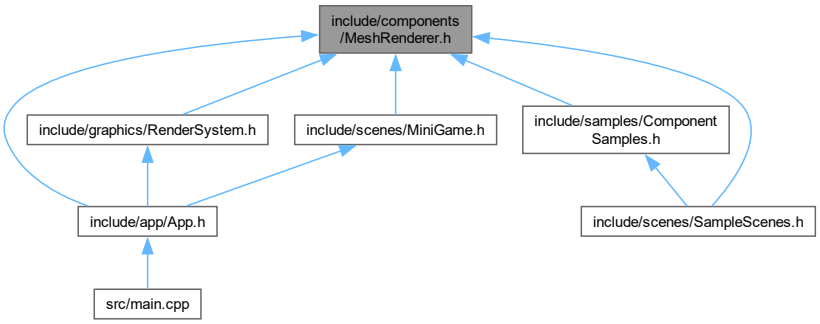
7.7 include/components/MeshRenderer.h ファイル

メッシュ描画コンポーネントの定義

#include <DirectXMath.h>
#include ”graphics/TextureManager.h”
MeshRenderer.h の依存先関係図:



被依存関係図:



クラス

- struct MeshRenderer
  オブジェクトの見た目（色・テクスチャ）を管理するデータコンポーネント

7.7.1 詳解

メッシュ描画コンポーネントの定義

著者

山内陽

日付

2024

バージョン

5.0

このファイルは 3D オブジェクトの「見た目」を制御するコンポーネントを定義します。

7.8 MeshRenderer.h

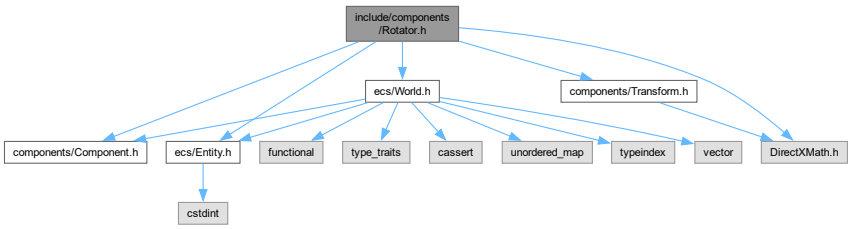
[詳解]

```
00001 #pragma once
00002 #include <DirectXMath.h>
00003 #include "graphics/TextureManager.h"
00004
00015
00068 struct MeshRenderer {
00094     DirectX::XMFLOAT3 color{ 0.3f, 0.7f, 1.0f };
00095
00116     TextureManager::TextureHandle texture = TextureManager::INVALID_TEXTURE;
00117
00135     DirectX::XMFLOAT2 uvOffset{ 0.0f, 0.0f };
00136
00154     DirectX::XMFLOAT2 uvScale{ 1.0f, 1.0f };
00155 };
```

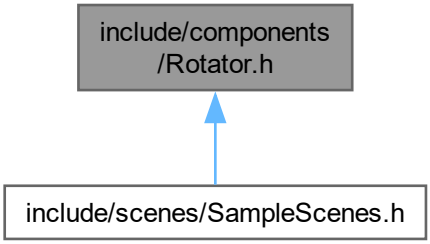
7.9 include/components/Rotator.h ファイル

自動回転Behaviour コンポーネントの定義

#include ”components/Component.h”
#include ”ecs/Entity.h”
#include ”ecs/World.h”
#include ”components/Transform.h”
#include <DirectXMath.h>
Rotator.h の依存先関係図:



被依存関係図:



クラス

- struct Rotator
  エンティティを自動的にY 軸中心で回転させるBehaviour コンポーネント

7.9.1 詳解

自動回転Behaviour コンポーネントの定義

著者

山内陽

日付

2024

バージョン

5.0

このファイルは、エンティティを自動的に回転させるBehaviour コンポーネントを定義します。Behaviour コンポーネントの実装例として、初学者の学習に最適です。

7.10 Rotator.h

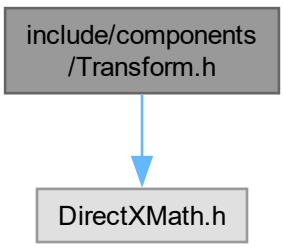
[詳解]

```
00001 #pragma once
00002 #include "components/Component.h"
00003 #include "ecs/Entity.h"
00004 #include "ecs/World.h"
00005 #include "components/Transform.h"
00006 #include <DirectXMath.h>
00007
00019
00073 struct Rotator : Behaviour {
00092     float speedDegY = 45.0f;
00093
00098     Rotator() = default;
00099
00111     explicit Rotator(float s) : speedDegY(s) {}
00112
00137     void OnUpdate(World& w, Entity self, float dt) override {
00138         // このエンティティの Transform を取得
00139         auto* t = w.TryGet<Transform>(self);
00140         if (!t) return; // Transform がなければ何もしない
00141
00142         // 回転値を更新（dt = デルタタイム = 前フレームからの経過秒数）
00143         t->rotation.y += speedDegY * dt;
00144
00145         // 360 度を超えたら正規化（見やすくするため、なくても OK）
00146         while (t->rotation.y >= 360.0f) t->rotation.y -= 360.0f;
00147         while (t->rotation.y < 0.0f) t->rotation.y += 360.0f;
00148     }
00149 };
```

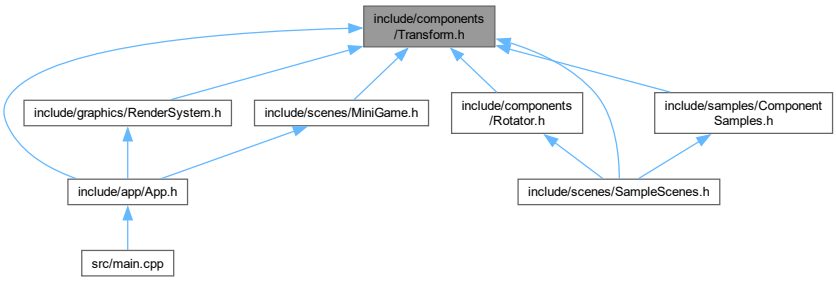
7.11 include/components/Transform.h ファイル

位置・回転・スケールコンポーネントの定義

#include <DirectXMath.h>
Transform.h の依存先関係図:



被依存関係図:



クラス

- struct Transform
  3D 空間におけるエンティティの位置・回転・スケールを管理するデータコンポーネント

7.11.1 詳解

位置・回転・スケールコンポーネントの定義

著者

山内陽

日付

2024

バージョン

4.0

このファイルは 3D 空間におけるエンティティの基本的な変換情報を管理する Transform コンポーネントを定義します。

7.12 Transform.h

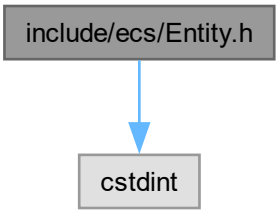
[詳解]

```
00001 #pragma once
00002 #include <DirectXMath.h>
00003
00015
00059 struct Transform {
00074     DirectX::XMFLOAT3 position{ 0, 0, 5 };
00075
00096     DirectX::XMFLOAT3 rotation{ 0, 0, 0 };
00097
00120     DirectX::XMFLOAT3 scale{ 1, 1, 1 };
00121 };
```

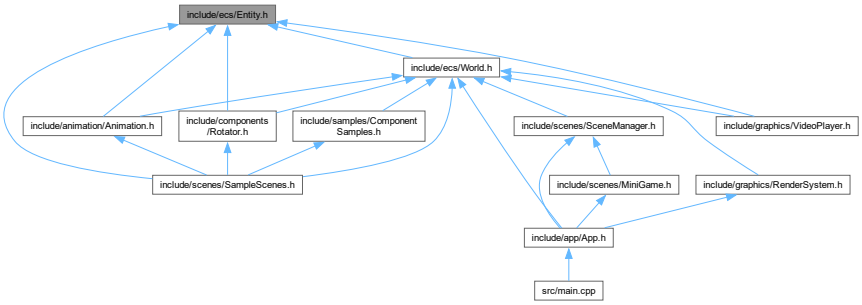
7.13 include/ecs/Entity.h ファイル

ECS アーキテクチャのエンティティ定義

#include <cstdint>
Entity.h の依存先関係図:



被依存関係図:



クラス

- struct Entity
  ゲーム世界に存在するオブジェクトを表す一意な識別子

7.13.1 詳解

ECS アーキテクチャのエンティティ定義

著者

山内陽

日付

2024

バージョン

5.0

このファイルはEntity Component System（ECS）アーキテクチャにおけるエンティティの基本定義を提供します。

7.14 Entity.h

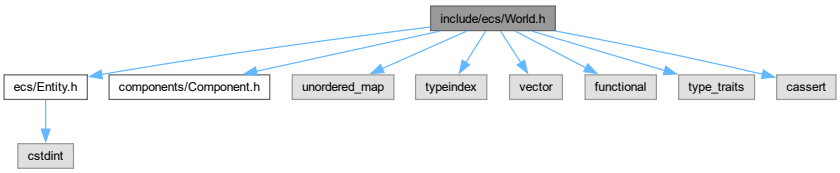
[詳解]

```
00001 #pragma once
00002 #include <cstdint>
00003
00015
00087 struct Entity {
00108     uint32_t id;
00109 };
```

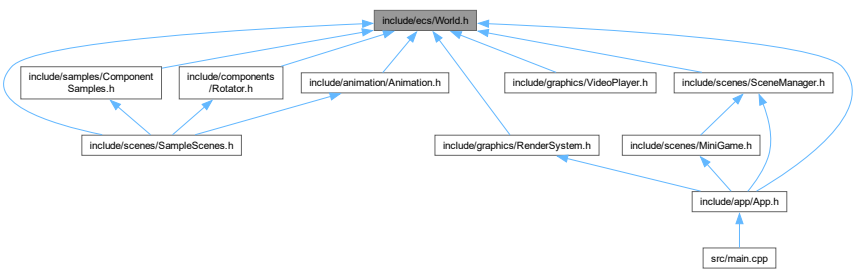
7.15 include/ecs/World.h ファイル

ECS ワールド管理システムとエンティティビルダーの定義

#include ”ecs/Entity.h”
#include ”components/Component.h”
#include <unordered_map>
#include <typeindex>
#include <vector>
#include <functional>
#include <type_traits>
#include <cassert>
World.h の依存先関係図:



被依存関係図:



クラス

- class EntityBuilder
  前方宣言
- class World
  ECS ワールド管理クラス - エンティティとコンポーネントのすべてを管理

7.15.1 詳解

ECS ワールド管理システムとエンティティビルダーの定義

著者

山内陽

日付

2024

バージョン

5.0

このファイルは、ECS アーキテクチャの中核となるWorld クラスと、エンティティを簡単に作成するためのEntityBuilder クラスを定義します。

7.16 World.h

[詳解]

```
00001 #pragma once
00002 #include "ecs/Entity.h"
00003 #include "components/Component.h"
00004 #include <unordered_map>
00005 #include <typeindex>
00006 #include <vector>
00007 #include <functional>
00008 #include <type_traits>
00009 #include <cassert>
00010
00022
00023 class World;
00024
00047 class EntityBuilder {
00048 public:
00054     EntityBuilder(World* world, Entity entity) : world_(world), entity_(entity) {}
00055
00073     template<typename T, typename... Args>
00074     EntityBuilder& With(Args&&... args);
00075
00082     Entity Build() { return entity_; }
00083
00096     operator Entity() const { return entity_; }
00097
00098 private:
00099     World* world_;
00100     Entity entity_;
00101 };
00102
00157 class World {
00158 public:
00176     Entity CreateEntity() {
00177         Entity e{ ++nextId_ };
```

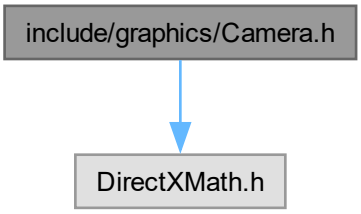
```
00178         alive_[e.id] = true;
00179         return e;
00180     }
00181
00201     EntityBuilder Create() {
00202         return EntityBuilder(this, CreateEntity());
00203     }
00204
00221     bool IsAlive(Entity e) const {
00222         auto it = alive_.find(e.id);
00223         return it != alive_.end() && it->second;
00224     }
00225
00246     void DestroyEntity(Entity e) {
00247         if (!IsAlive(e)) return;
00248         alive_[e.id] = false;
00249         for (auto& er : erasers_) er(e);
00250         for (size_t i = 0; i < behaviours_.size(); ++i)
00251             if (behaviours_[i].e.id == e.id) { behaviours_.erase(behaviours_.begin() + i); --i; }
00252     }
00253
00277     template<class T, class...Args>
00278     T& Add(Entity e, Args&&...args) {
00279         assert(IsAlive(e) && "Add on dead entity");
00280         auto& s = getStore<T>();
00281         T& obj = s.map[e.id] = T{ std::forward<Args>(args)... };
00282         registerBehaviour<T>(e, &obj);
00283         return obj;
00284     }
00285
00307     template<class T>
00308     T* TryGet(Entity e) {
00309         auto itS = stores_.find(std::type_index(typeid(T)));
00310         if (itS == stores_.end()) return nullptr;
00311         auto* s = static_cast<Store<T>*>(itS->second);
00312         auto it = s->map.find(e.id);
00313         if (it == s->map.end()) return nullptr;
00314         return &it->second;
00315     }
00316
00329     template<class T>
00330     bool Remove(Entity e) {
00331         auto itS = stores_.find(std::type_index(typeid(T)));
00332         if (itS == stores_.end()) return false;
00333         auto* s = static_cast<Store<T>*>(itS->second);
00334         unregisterBehaviour<T>(e);
00335         return s->map.erase(e.id) > 0;
00336     }
00337
00361     template<class T, class F>
00362     void ForEach(F&& fn) {
00363         auto itS = stores_.find(std::type_index(typeid(T)));
00364         if (itS == stores_.end()) return;
00365         auto* s = static_cast<Store<T>*>(itS->second);
00366         for (auto it = s->map.begin(); it != s->map.end(); ++it) {
00367             Entity e{ it->first };
00368             if (!IsAlive(e)) continue;
00369             fn(e, it->second);
00370         }
00371     }
00372
00394     void Tick(float dt) {
00395         // イテレーション中の削除に対応するため、インデックスベースのループを使用
00396         for (size_t i = 0; i < behaviours_.size(); ++i) {
00397             auto& it = behaviours_[i];
00398             if (!IsAlive(it.e)) continue;
00399             if (!it.started) { it.b->OnStart(*this, it.e); it.started = true; }
00400             it.b->OnUpdate(*this, it.e, dt);
00401         }
00402     }
00403
00404 private:
00410     struct IStore {
00411         virtual ~IStore() = default;
00412         virtual void Erase(Entity) = 0;
00413     };
00414
00421     template<class T>
00422     struct Store : IStore {
00423         std::unordered_map<uint32_t, T> map;
00424         void Erase(Entity e) override { map.erase(e.id); }
00425     };
00426
00428     template<class T>
00429     Store<T>& getStore() {
00430         auto key = std::type_index(typeid(T));
00431         auto it = stores_.find(key);
```

```
00432         if (it == stores_.end()) {
00433             auto* s = new Store<T>();
00434             stores_[key] = s;
00435             erasers_.push_back([s](Entity e) { s->Erase(e); });
00436             return *s;
00437         }
00438         return *static_cast<Store<T>*>(it->second);
00439     }
00440
00442     template<class TDerived>
00443     typename std::enable_if<std::is_base_of<Behaviour, TDerived>::value>::type
00444         registerBehaviour(Entity e, TDerived* obj) {
00445         behaviours_.push_back({ e, obj, false });
00446     }
00447     template<class TDerived>
00448     typename std::enable_if<!std::is_base_of<Behaviour, TDerived>::value>::type
00449         registerBehaviour(Entity, TDerived*) {}
00450
00452     template<class TDerived>
00453     typename std::enable_if<std::is_base_of<Behaviour, TDerived>::value>::type
00454         unregisterBehaviour(Entity e) {
00455         for (size_t i = 0; i < behaviours_.size(); ++i) {
00456             if (behaviours_[i].e.id == e.id) { behaviours_.erase(behaviours_.begin() + i); break; }
00457         }
00458     }
00459     template<class TDerived>
00460     typename std::enable_if<!std::is_base_of<Behaviour, TDerived>::value>::type
00461         unregisterBehaviour(Entity) {}
00462
00468     struct BEntry {
00469         Entity e;
00470         Behaviour* b;
00471         bool started = false;
00472     };
00473
00474     uint32_t nextId_ = 0;
00475     std::unordered_map<uint32_t, bool> alive_;
00476     std::unordered_map<std::type_index, IStore*> stores_;
00477     std::vector<std::function<void(Entity)» erasers_;
00478     std::vector<BEntry> behaviours_;
00479
00480     friend class EntityBuilder;
00481 };
00482
00490 template<typename T, typename... Args>
00491 EntityBuilder& EntityBuilder::With(Args&&... args) {
00492     world_->Add<T>(entity_, std::forward<Args>(args)...);
00493     return *this;
00494 }
```

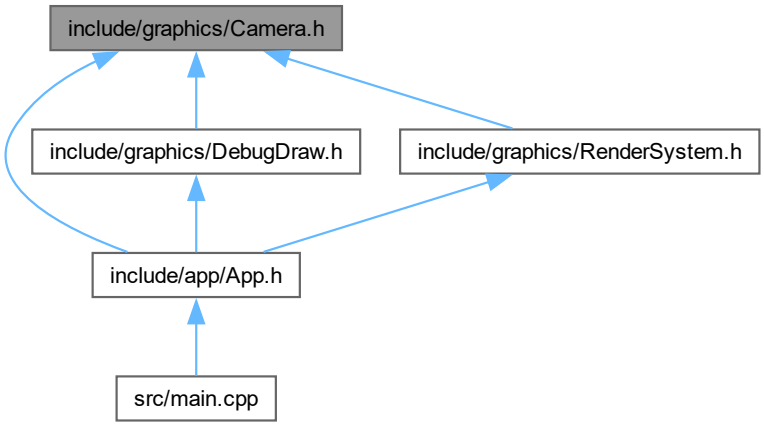
7.17 include/graphics/Camera.h ファイル

3D カメラシステムの定義

#include <DirectXMath.h>
Camera.h の依存先関係図:



被依存関係図:



クラス

- struct Camera
  3D 空間のカメラ（ビュー・プロジェクション行列）を管理

7.17.1 詳解

3D カメラシステムの定義

著者

山内陽

日付

2024

バージョン

5.0

このファイルは 3D 空間のカメラ（視点）を管理する構造体を定義します。

7.18 Camera.h

[詳解]

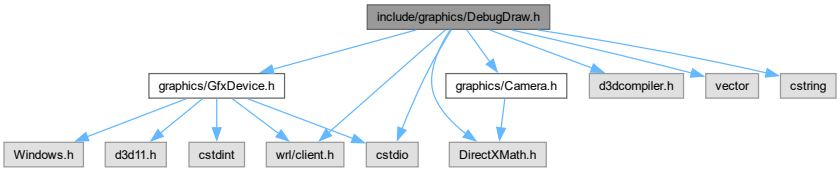
```
00001 #pragma once
00002 #include <DirectXMath.h>
00003
00014
00057 struct Camera {
00058     DirectX::XMMATRIX View;
00059     DirectX::XMMATRIX Proj;
00060
00061     DirectX::XMFLOAT3 position;
00062     DirectX::XMFLOAT3 target;
00063     DirectX::XMFLOAT3 up;
00064
00065     float fovY;
00066     float aspect;
00067     float nearZ;
00068     float farZ;
00069
00103     static Camera LookAtLH(
00104         float fovY, float aspect, float znear, float zfar,
00105         DirectX::XMFLOAT3 eye, DirectX::XMFLOAT3 at, DirectX::XMFLOAT3 upVec)
00106     {
00107         Camera c;
00108         c.position = eye;
00109         c.target = at;
00110         c.up = upVec;
00111         c.fovY = fovY;
00112         c.aspect = aspect;
00113         c.nearZ = znear;
00114         c.farZ = zfar;
00115
00116         c.View = DirectX::XMMatrixLookAtLH(
00117             DirectX::XMLoadFloat3(&eye),
00118             DirectX::XMLoadFloat3(&at),
00119             DirectX::XMLoadFloat3(&upVec));
00120         c.Proj = DirectX::XMMatrixPerspectiveFovLH(fovY, aspect, znear, zfar);
00121         return c;
00122     }
00123
00137     void Update() {
00138         View = DirectX::XMMatrixLookAtLH(
00139             DirectX::XMLoadFloat3(&position),
00140             DirectX::XMLoadFloat3(&target),
00141             DirectX::XMLoadFloat3(&up));
00142     }
00143
00164     void Orbit(float deltaYaw, float deltaPitch) {
00165         // 現在の位置からターゲットへのベクトル
00166         DirectX::XMVECTOR posVec = DirectX::XMLoadFloat3(&position);
00167         DirectX::XMVECTOR targetVec = DirectX::XMLoadFloat3(&target);
00168         DirectX::XMVECTOR toTarget = DirectX::XMVectorSubtract(targetVec, posVec);
00169
00170         float radius = DirectX::XMVectorGetX(DirectX::XMVector3Length(toTarget));
00171
00172         // 球面座標系で回転
00173         DirectX::XMMATRIX rotY = DirectX::XMMatrixRotationY(deltaYaw);
00174         DirectX::XMMATRIX rotX = DirectX::XMMatrixRotationAxis(
00175             DirectX::XMVector3Cross(toTarget, DirectX::XMLoadFloat3(&up)),
00176             deltaPitch);
00177
00178         DirectX::XMVECTOR newDir = DirectX::XMVector3TransformNormal(toTarget, rotY);
00179         newDir = DirectX::XMVector3TransformNormal(newDir, rotX);
00180         newDir = DirectX::XMVector3Normalize(newDir);
00181
00182         DirectX::XMVECTOR newPos = DirectX::XMVectorAdd(
00183             targetVec,
00184             DirectX::XMVectorScale(newDir, -radius));
00185
00186         DirectX::XMStoreFloat3(&position, newPos);
00187         Update();
00188     }
00189
00205     void Zoom(float delta) {
00206         fovY += delta;
00207         // 視野角を制限（22.5 度〜90 度）
00208         if (fovY < DirectX::XM_PIDIV4 * 0.5f) fovY = DirectX::XM_PIDIV4 * 0.5f;
00209         if (fovY > DirectX::XM_PIDIV2) fovY = DirectX::XM_PIDIV2;
00210
00211         Proj = DirectX::XMMatrixPerspectiveFovLH(fovY, aspect, nearZ, farZ);
00212     }
00213 };
```

7.19 include/graphics/DebugDraw.h ファイル

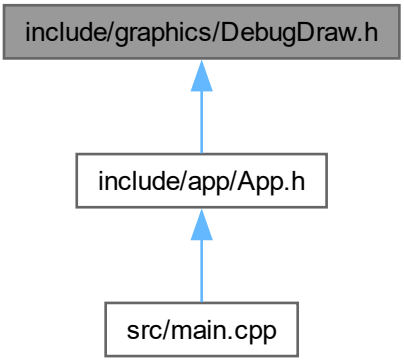
デバッグ用の線描画システム

```
#include "graphics/GfxDevice.h"
#include "graphics/Camera.h"
#include <d3dcompiler.h>
#include <DirectXMath.h>
#include <wrl/client.h>
#include <vector>
#include <cstring>
#include <cstdio>
```

DebugDraw.h の依存先関係図:



被依存関係図:



クラス

- class DebugDraw
  デバッグ用の線描画システム
- struct DebugDraw::Line
  線分の定義（開始点、終了点、色）

7.19.1 詳解

デバッグ用の線描画システム

著者

山内陽

日付

2024

バージョン

5.0

デバッグ時にグリッドや軸、任意の線を描画するためのシステムです。Release ビルドでは使用されません。

7.20 DebugDraw.h

[詳解]

```
00001
00012 #pragma once
00013 #include "graphics/GfxDevice.h"
00014 #include "graphics/Camera.h"
00015 #include <d3dcompiler.h>
00016 #include <DirectXMath.h>
00017 #include <wrl/client.h>
00018 #include <vector>
00019 #include <cstring>
00020 #include <cstdio>
00021
00022 #pragma comment(lib, "d3dcompiler.lib")
00023
00065 class DebugDraw {
00066 public:
00071     struct Line {
00072         DirectX::XMFLOAT3 start;
00073         DirectX::XMFLOAT3 end;
00074         DirectX::XMFLOAT3 color;
00075     };
00076
00082     bool Init(GfxDevice& gfx) {
00083         // シェーダーのコンパイル
00084         const char* VS = R"(
00085             cbuffer CB : register(b0) { float4x4 gVP; };
00086             struct VSIn { float3 pos : POSITION; float3 col : COLOR; };
00087             struct VSOut { float4 pos : SV_POSITION; float3 col : COLOR; };
00088             VSOut main(VSIn i){
00089                 VSOut o;
00090                 o.pos = mul(float4(i.pos, 1), gVP);
00091                 o.col = i.col;
00092                 return o;
00093             }
00094         )";
00095
00096         const char* PS = R"(
00097             struct VSOut { float4 pos : SV_POSITION; float3 col : COLOR; };
00098             float4 main(VSOut i) : SV_Target { return float4(i.col, 1); }
00099         )";
00100
00101         Microsoft::WRL::ComPtr<ID3DBlob> vsb, psb, err;
00102         HRESULT hr = D3DCompile(VS, strlen(VS), nullptr, nullptr, nullptr, "main", "vs_5_0", 0, 0, vsb.GetAddressOf(),
    err.GetAddressOf());
00103         if (FAILED(hr)) {
00104             if (err) {
00105                 char msg[512];
```

```
00106                 sprintf_s(msg, "DebugDraw VS compile error:\n%s", (char*)err->GetBufferPointer());
00107                 MessageBoxA(nullptr, msg, "Shader Error", MB_OK | MB_ICONERROR);
00108             }
00109             return false;
00110         }
00111
00112         hr = D3DCompile(PS, strlen(PS), nullptr, nullptr, nullptr, "main", "ps_5_0", 0, 0, psb.GetAddressOf(),
      err.ReleaseAndGetAddressOf());
00113         if (FAILED(hr)) {
00114             if (err) {
00115                 char msg[512];
00116                 sprintf_s(msg, "DebugDraw PS compile error:\n%s", (char*)err->GetBufferPointer());
00117                 MessageBoxA(nullptr, msg, "Shader Error", MB_OK | MB_ICONERROR);
00118             }
00119             return false;
00120         }
00121
00122         if (FAILED(gfx.Dev()->CreateVertexShader(vsb->GetBufferPointer(), vsb->GetBufferSize(), nullptr,
      vs_.GetAddressOf()))) {
00123             MessageBoxA(nullptr, "Failed to create debug vertex shader", "Shader Error", MB_OK | MB_ICONERROR);
00124             return false;
00125         }
00126
00127         if (FAILED(gfx.Dev()->CreatePixelShader(psb->GetBufferPointer(), psb->GetBufferSize(), nullptr,
      ps_.GetAddressOf()))) {
00128             MessageBoxA(nullptr, "Failed to create debug pixel shader", "Shader Error", MB_OK | MB_ICONERROR);
00129             return false;
00130         }
00131
00132         // 入力レイアウト
00133         D3D11_INPUT_ELEMENT_DESC il[] = {
00134             { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
      D3D11_INPUT_PER_VERTEX_DATA, 0 },
00135             { "COLOR",    0, DXGI_FORMAT_R32G32B32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT,
      D3D11_INPUT_PER_VERTEX_DATA, 0 }
00136         };
00137         if (FAILED(gfx.Dev()->CreateInputLayout(il, 2, vsb->GetBufferPointer(), vsb->GetBufferSize(),
      layout_.GetAddressOf()))) {
00138             MessageBoxA(nullptr, "Failed to create debug input layout", "Shader Error", MB_OK | MB_ICONERROR);
00139             return false;
00140         }
00141
00142         // 定数バッファ
00143         D3D11_BUFFER_DESC cbd{};
00144         cbd.ByteWidth = sizeof(DirectX::XMMATRIX);
00145         cbd.Usage = D3D11_USAGE_DEFAULT;
00146         cbd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
00147         if (FAILED(gfx.Dev()->CreateBuffer(&cbd, nullptr, cb_.GetAddressOf()))) {
00148             MessageBoxA(nullptr, "Failed to create debug constant buffer", "Buffer Error", MB_OK | MB_ICONERROR);
00149             return false;
00150         }
00151
00152         // 動的頂点バッファ（最大 10000 線分）
00153         maxLines_ = 10000;
00154         D3D11_BUFFER_DESC vbd{};
00155         vbd.ByteWidth = (UINT)(maxLines_ * 2 * sizeof(Vertex)); // 1 線分 = 2 頂点
00156         vbd.Usage = D3D11_USAGE_DYNAMIC;
00157         vbd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
00158         vbd.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
00159         if (FAILED(gfx.Dev()->CreateBuffer(&vbd, nullptr, vb_.GetAddressOf()))) {
00160             MessageBoxA(nullptr, "Failed to create debug vertex buffer", "Buffer Error", MB_OK | MB_ICONERROR);
00161             return false;
00162         }
00163
00164         return true;
00165     }
00166
00173     void AddLine(const DirectX::XMFLOAT3& start, const DirectX::XMFLOAT3& end, const DirectX::XMFLOAT3&
      color) {
00174         lines_.push_back({ start, end, color });
00175     }
00176
00183     void DrawGrid(float size = 10.0f, int divisions = 10, const DirectX::XMFLOAT3& color = {0.5f, 0.5f, 0.5f}) {
00184         float step = size / divisions;
00185         float halfSize = size * 0.5f;
00186
00187         // X-Z 平面のグリッド
00188         for (int i = 0; i <= divisions; ++i) {
00189             float pos = -halfSize + i * step;
00190
00191             // Z 軸に平行な線（X 方向に並ぶ）
00192             AddLine(
00193                 DirectX::XMFLOAT3{-halfSize, 0, pos},
00194                 DirectX::XMFLOAT3{ halfSize, 0, pos},
00195                 color
00196             );
00197
```

```
00198             // X 軸に平行な線（Z 方向に並ぶ）
00199             AddLine(
00200                 DirectX::XMFLOAT3{pos, 0, -halfSize},
00201                 DirectX::XMFLOAT3{pos, 0,  halfSize},
00202                 color
00203             );
00204         }
00205     }
00206
00214     void DrawAxes(float length = 5.0f) {
00215         // X 軸（赤）
00216         AddLine(
00217             DirectX::XMFLOAT3{0, 0, 0},
00218             DirectX::XMFLOAT3{length, 0, 0},
00219             DirectX::XMFLOAT3{1, 0, 0}
00220         );
00221
00222         // Y 軸（緑）
00223         AddLine(
00224             DirectX::XMFLOAT3{0, 0, 0},
00225             DirectX::XMFLOAT3{0, length, 0},
00226             DirectX::XMFLOAT3{0, 1, 0}
00227         );
00228
00229         // Z 軸（青）
00230         AddLine(
00231             DirectX::XMFLOAT3{0, 0, 0},
00232             DirectX::XMFLOAT3{0, 0, length},
00233             DirectX::XMFLOAT3{0, 0, 1}
00234         );
00235     }
00236
00242     void Render(GfxDevice& gfx, const Camera& cam) {
00243         if (lines_.empty()) return;
00244
00245         // 頂点データを構築
00246         std::vector<Vertex> vertices;
00247         vertices.reserve(lines_.size() * 2);
00248
00249         for (const auto& line : lines_) {
00250             vertices.push_back({ line.start, line.color });
00251             vertices.push_back({ line.end, line.color });
00252         }
00253
00254         // 頂点バッファを更新
00255         D3D11_MAPPED_SUBRESOURCE mapped;
00256         if (SUCCEEDED(gfx.Ctx()->Map(vb_.Get(), 0, D3D11_MAP_WRITE_DISCARD, 0, &mapped))) {
00257             memcpy(mapped.pData, vertices.data(), vertices.size() * sizeof(Vertex));
00258             gfx.Ctx()->Unmap(vb_.Get(), 0);
00259         }
00260
00261         // パイプライン設定
00262         gfx.Ctx()->IASetInputLayout(layout_.Get());
00263         gfx.Ctx()->VSSetShader(vs_.Get(), nullptr, 0);
00264         gfx.Ctx()->PSSetShader(ps_.Get(), nullptr, 0);
00265         gfx.Ctx()->VSSetConstantBuffers(0, 1, cb_.GetAddressOf());
00266
00267         UINT stride = sizeof(Vertex);
00268         UINT offset = 0;
00269         gfx.Ctx()->IASetVertexBuffers(0, 1, vb_.GetAddressOf(), &stride, &offset);
00270         gfx.Ctx()->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_LINELIST);
00271
00272         // 定数バッファ更新（ワールド行列は単位行列）
00273         DirectX::XMMATRIX VP = DirectX::XMMatrixTranspose(cam.View * cam.Proj);
00274         gfx.Ctx()->UpdateSubresource(cb_.Get(), 0, nullptr, &VP, 0, 0);
00275
00276         // 描画
00277         gfx.Ctx()->Draw((UINT)vertices.size(), 0);
00278     }
00279
00287     void Clear() {
00288         lines_.clear();
00289     }
00290
00294     ~DebugDraw() {
00295         vs_.Reset();
00296         ps_.Reset();
00297         layout_.Reset();
00298         cb_.Reset();
00299         vb_.Reset();
00300     }
00301
00302 private:
00307     struct Vertex {
00308         DirectX::XMFLOAT3 pos;
00309         DirectX::XMFLOAT3 col;
00310     };
```

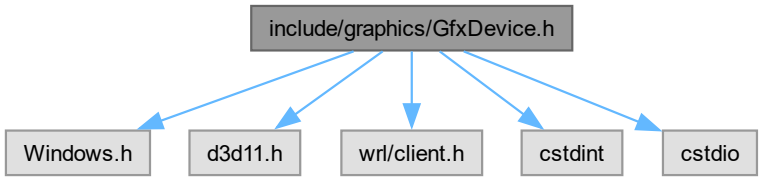
```
00311
00312     Microsoft::WRL::ComPtr<ID3D11VertexShader> vs_;
00313     Microsoft::WRL::ComPtr<ID3D11PixelShader> ps_;
00314     Microsoft::WRL::ComPtr<ID3D11InputLayout> layout_;
00315     Microsoft::WRL::ComPtr<ID3D11Buffer> cb_;
00316     Microsoft::WRL::ComPtr<ID3D11Buffer> vb_;
00317
00318     std::vector<Line> lines_;
00319     size_t maxLines_;
00320 };
```

7.21 include/graphics/GfxDevice.h ファイル

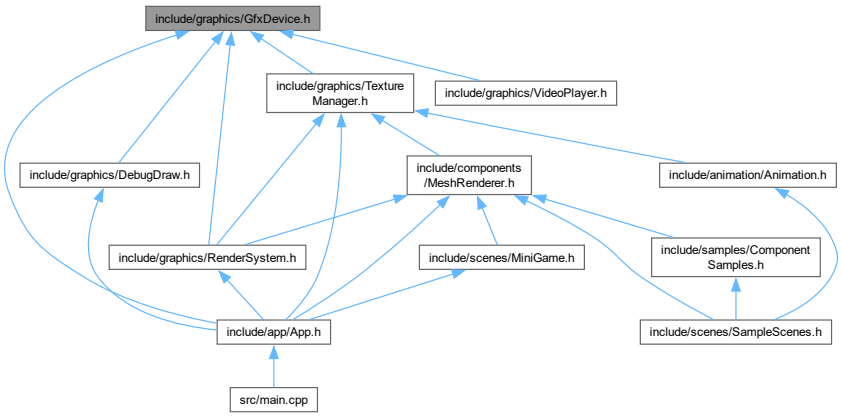
DirectX11 デバイス管理クラス

```
#include <Windows.h>
#include <d3d11.h>
#include <wrl/client.h>
#include <cstdint>
#include <cstdio>
```

GfxDevice.h の依存先関係図:



被依存関係図:



クラス

- class GfxDevice
  DirectX11 デバイス管理クラス

マクロ定義

- #define WIN32_LEAN_AND_MEAN
- #define NOMINMAX

7.21.1 詳解

DirectX11 デバイス管理クラス

DirectX11 の初期化、デバイス・コンテキストの管理、描画フレームの制御を行います

7.21.2 マクロ定義詳解

7.21.2.1 NOMINMAX

#define NOMINMAX

7.21.2.2 WIN32_LEAN_AND_MEAN

#define WIN32_LEAN_AND_MEAN

7.22 GfxDevice.h

[詳解]

```
00001
00006 #pragma once
00007 #define WIN32_LEAN_AND_MEAN
00008 #define NOMINMAX
00009 #include <Windows.h>
00010 #include <d3d11.h>
00011 #include <wrl/client.h>
00012 #include <cstdint>
00013 #include <cstdio>
00014
00022 class GfxDevice {
00023 public:
00031     bool Init(HWND hwnd, uint32_t w, uint32_t h) {
00032         width_ = w;
00033         height_ = h;
00034
00035         DXGI_SWAP_CHAIN_DESC sd{};
00036         sd.BufferCount = 2;
00037         sd.BufferDesc.Width = w;
00038         sd.BufferDesc.Height = h;
00039         sd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
00040         sd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
00041         sd.OutputWindow = hwnd;
00042         sd.SampleDesc.Count = 1;
00043         sd.Windowed = TRUE;
00044         sd.SwapEffect = DXGI_SWAP_EFFECT_DISCARD;
00045
00046         UINT flags = 0;
00047 #if defined(_DEBUG)
00048         flags |= D3D11_CREATE_DEVICE_DEBUG;
00049 #endif
00050         D3D_FEATURE_LEVEL fl;
00051         HRESULT hr = D3D11CreateDeviceAndSwapChain(
00052             nullptr, D3D_DRIVER_TYPE_HARDWARE, nullptr, flags,
00053             nullptr, 0, D3D11_SDK_VERSION,
00054             &sd,
00055             swap_.ReleaseAndGetAddressOf(),
00056             device_.ReleaseAndGetAddressOf(),
00057             &fl,
```

```
00058           context_.ReleaseAndGetAddressOf());
00059
00060       if (FAILED(hr)) {
00061           // エラーの詳細をログ出力
00062           char errorMsg[256];
00063           sprintf_s(errorMsg,
00064               "Failed to create D3D11 device.\nHRESULT: 0x%08X\n"
00065               "Please check:\n"
00066               "- DirectX 11 is installed\n"
00067               "- Graphics drivers are up to date",
00068               hr);
00069           MessageBoxA(nullptr, errorMsg, "DirectX Error", MB_OK | MB_ICONERROR);
00070           return false;
00071       }
00072
00073       return createBackbufferResources();
00074   }
00075
00083   void BeginFrame(float r = 0.1f, float g = 0.1f, float b = 0.12f, float a = 1.0f) {
00084       float c[4] = { r, g, b, a };
00085       context_->OMSetRenderTargets(1, rtv_.GetAddressOf(), dsv_.Get());
00086       context_->ClearRenderTargetView(rtv_.Get(), c);
00087       context_->ClearDepthStencilView(dsv_.Get(), D3D11_CLEAR_DEPTH | D3D11_CLEAR_STENCIL, 1.0f, 0);
00088
00089       D3D11_VIEWPORT vp{};
00090       vp.Width = static_cast<FLOAT>(width_);
00091       vp.Height = static_cast<FLOAT>(height_);
00092       vp.MinDepth = 0.0f;
00093       vp.MaxDepth = 1.0f;
00094       vp.TopLeftX = 0;
00095       vp.TopLeftY = 0;
00096       context_->RSSetViewports(1, &vp);
00097   }
00098
00102   void EndFrame() {
00103       swap_->Present(1, 0);
00104   }
00105
00110   ID3D11Device* Dev() const { return device_.Get(); }
00111
00116   ID3D11DeviceContext* Ctx() const { return context_.Get(); }
00117
00122   uint32_t Width() const { return width_; }
00123
00128   uint32_t Height() const { return height_; }
00129
00133   ~GfxDevice() {
00134       // ComPtr は自動で解放されるが、念のため明示的にリセット
00135       dsv_.Reset();
00136       rtv_.Reset();
00137       swap_.Reset();
00138       context_.Reset();
00139       device_.Reset();
00140   }
00141
00142 private:
00147   bool createBackbufferResources() {
00148       Microsoft::WRL::ComPtr<ID3D11Texture2D> back;
00149       HRESULT hr = swap_->GetBuffer(0, __uuidof(ID3D11Texture2D), (void**)back.GetAddressOf());
00150       if (FAILED(hr)) {
00151           MessageBoxA(nullptr, "Failed to get back buffer", "DirectX Error", MB_OK | MB_ICONERROR);
00152           return false;
00153       }
00154
00155       hr = device_->CreateRenderTargetView(back.Get(), nullptr, rtv_.ReleaseAndGetAddressOf());
00156       if (FAILED(hr)) {
00157           MessageBoxA(nullptr, "Failed to create render target view", "DirectX Error", MB_OK | MB_ICONERROR);
00158           return false;
00159       }
00160
00161       // 深度ステンシルバッファ
00162       D3D11_TEXTURE2D_DESC td{};
00163       td.Width = width_;
00164       td.Height = height_;
00165       td.MipLevels = 1;
00166       td.ArraySize = 1;
00167       td.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;
00168       td.SampleDesc.Count = 1;
00169       td.Usage = D3D11_USAGE_DEFAULT;
00170       td.BindFlags = D3D11_BIND_DEPTH_STENCIL;
00171
00172       Microsoft::WRL::ComPtr<ID3D11Texture2D> depth;
00173       hr = device_->CreateTexture2D(&td, nullptr, depth.GetAddressOf());
00174       if (FAILED(hr)) {
00175           MessageBoxA(nullptr, "Failed to create depth stencil texture", "DirectX Error", MB_OK | MB_ICONERROR);
00176           return false;
00177       }
```

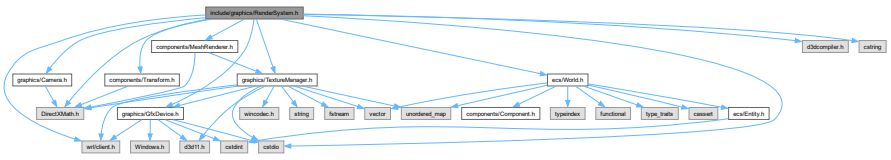
```
00178
00179         hr = device_->CreateDepthStencilView(depth.Get(), nullptr, dsv_.ReleaseAndGetAddressOf());
00180         if (FAILED(hr)) {
00181             MessageBoxA(nullptr, "Failed to create depth stencil view", "DirectX Error", MB_OK | MB_ICONERROR);
00182             return false;
00183         }
00184
00185         return true;
00186     }
00187
00188     // メンバ変数
00189     uint32_t width_ = 0;
00190     uint32_t height_ = 0;
00191     Microsoft::WRL::ComPtr<ID3D11Device> device_;
00192     Microsoft::WRL::ComPtr<ID3D11DeviceContext> context_;
00193     Microsoft::WRL::ComPtr<IDXGISwapChain> swap_;
00194     Microsoft::WRL::ComPtr<ID3D11RenderTargetView> rtv_;
00195     Microsoft::WRL::ComPtr<ID3D11DepthStencilView> dsv_;
00196 };
```

7.23 include/graphics/RenderSystem.h ファイル

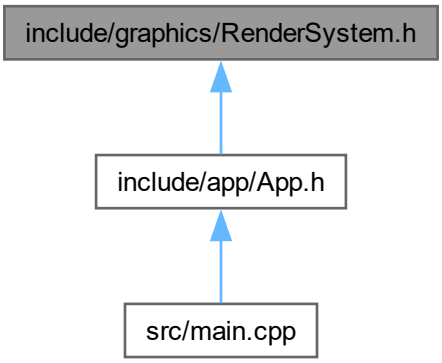
テクスチャ対応レンダリングシステム

```
#include "graphics/GfxDevice.h"
#include "graphics/Camera.h"
#include "ecs/World.h"
#include "components/Transform.h"
#include "components/MeshRenderer.h"
#include "graphics/TextureManager.h"
#include <d3dcompiler.h>
#include <DirectXMath.h>
#include <wrl/client.h>
#include <cstring>
#include <cstdio>
```

RenderSystem.h の依存先関係図:



被依存関係図:



クラス

- struct RenderSystem
  テクスチャ対応レンダリングシステム
- struct RenderSystem::VSConstants
  頂点シェーダー定数バッファ
- struct RenderSystem::PSConstants
  ピクセルシェーダー定数バッファ

7.23.1 詳解

テクスチャ対応レンダリングシステム

著者

山内陽

日付

2024

バージョン

5.0

ECS ワールド内のMeshRenderer コンポーネントを持つエンティティを自動的に描画するレンダリングシステムです。

7.24 RenderSystem.h

[詳解]

```
00001
00012 #pragma once
00013 #include "graphics/GfxDevice.h"
00014 #include "graphics/Camera.h"
00015 #include "ecs/World.h"
00016 #include "components/Transform.h"
00017 #include "components/MeshRenderer.h"
00018 #include "graphics/TextureManager.h"
00019 #include <d3dcompiler.h>
00020 #include <DirectXMath.h>
00021 #include <wrl/client.h>
00022 #include <cstring>
00023 #include <cstdio>
00024
00025 #pragma comment(lib, "d3dcompiler.lib")
00026
00056 struct RenderSystem {
00057     // パイプラインオブジェクト
00058     Microsoft::WRL::ComPtr<ID3D11VertexShader> vs_;
00059     Microsoft::WRL::ComPtr<ID3D11PixelShader> ps_;
00060     Microsoft::WRL::ComPtr<ID3D11InputLayout> layout_;
00061     Microsoft::WRL::ComPtr<ID3D11Buffer> cb_;
00062     Microsoft::WRL::ComPtr<ID3D11Buffer> vb_;
00063     Microsoft::WRL::ComPtr<ID3D11Buffer> ib_;
00064     Microsoft::WRL::ComPtr<ID3D11RasterizerState> rasterState_;
00065     Microsoft::WRL::ComPtr<ID3D11SamplerState> samplerState_;
00066     UINT indexCount_ = 0;
00067
00068     TextureManager* texManager_ = nullptr;
00069
00074     struct VSConstants {
00075         DirectX::XMMATRIX WVP;
00076         DirectX::XMFLOAT4 uvTransform;
00077     };
00078
00083     struct PSConstants {
00084         DirectX::XMFLOAT4 color;
00085         float useTexture;
00086         float padding[3];
00087     };
00088
00089     Microsoft::WRL::ComPtr<ID3D11Buffer> psCb_;
00090
00094     ~RenderSystem() {
00095         vs_.Reset();
00096         ps_.Reset();
00097         layout_.Reset();
00098         cb_.Reset();
00099         psCb_.Reset();
00100         vb_.Reset();
00101         ib_.Reset();
00102         rasterState_.Reset();
00103         samplerState_.Reset();
00104     }
00105
00112     bool Init(GfxDevice& gfx, TextureManager& texMgr) {
00113         texManager_ = &texMgr;
00114
00115         // テクスチャ対応シェーダー
00116         const char* VS = R"(
00117             cbuffer CB : register(b0) {
00118                 float4x4 gWVP;
00119                 float4 gUVTransform; // xy=offset, zw=scale
00120             };
00121             struct VSIn {
00122                 float3 pos : POSITION;
00123                 float2 tex : TEXCOORD;
00124             };
00125             struct VSOut {
00126                 float4 pos : SV_POSITION;
00127                 float2 tex : TEXCOORD;
00128             };
00129             VSOut main(VSIn i){
00130                 VSOut o;
00131                 o.pos = mul(float4(i.pos,1), gWVP);
00132                 o.tex = i.tex * gUVTransform.zw + gUVTransform.xy;
00133                 return o;
00134             }
00135         )";
00136
00137         const char* PS = R"(
00138             cbuffer CB : register(b0) {
```

```
00139               float4 gColor;
00140               float gUseTexture;
00141               float3 padding;
00142           };
00143           Texture2D gTexture : register(t0);
00144           SamplerState gSampler : register(s0);
00145           struct VSOut {
00146               float4 pos : SV_POSITION;
00147               float2 tex : TEXCOORD;
00148           };
00149           float4 main(VSOut i) : SV_Target {
00150               if (gUseTexture > 0.5) {
00151                   return gTexture.Sample(gSampler, i.tex) * gColor;
00152               }
00153               return gColor;
00154           }
00155       )";
00156
00157       Microsoft::WRL::ComPtr<ID3DBlob> vsb, psb, err;
00158       HRESULT hr = D3DCompile(VS, strlen(VS), nullptr, nullptr, nullptr, "main", "vs_5_0", 0, 0, vsb.GetAddressOf(),
      err.GetAddressOf());
00159       if (FAILED(hr)) {
00160           if (err) {
00161               char msg[512];
00162               sprintf_s(msg, "Vertex Shader compile error:\n%s", (char*)err->GetBufferPointer());
00163               MessageBoxA(nullptr, msg, "Shader Error", MB_OK | MB_ICONERROR);
00164           }
00165           return false;
00166       }
00167
00168       hr = D3DCompile(PS, strlen(PS), nullptr, nullptr, nullptr, "main", "ps_5_0", 0, 0, psb.GetAddressOf(),
      err.ReleaseAndGetAddressOf());
00169       if (FAILED(hr)) {
00170           if (err) {
00171               char msg[512];
00172               sprintf_s(msg, "Pixel Shader compile error:\n%s", (char*)err->GetBufferPointer());
00173               MessageBoxA(nullptr, msg, "Shader Error", MB_OK | MB_ICONERROR);
00174           }
00175           return false;
00176       }
00177
00178       if (FAILED(gfx.Dev()->CreateVertexShader(vsb->GetBufferPointer(), vsb->GetBufferSize(), nullptr,
      vs_.GetAddressOf()))) {
00179           return false;
00180       }
00181
00182       if (FAILED(gfx.Dev()->CreatePixelShader(psb->GetBufferPointer(), psb->GetBufferSize(), nullptr,
      ps_.GetAddressOf()))) {
00183           return false;
00184       }
00185
00186       // 入力レイアウト (Position + TexCoord)
00187       D3D11_INPUT_ELEMENT_DESC il[] = {
00188           { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
      D3D11_INPUT_PER_VERTEX_DATA, 0 },
00189           { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT,    0, D3D11_APPEND_ALIGNED_ELEMENT,
      D3D11_INPUT_PER_VERTEX_DATA, 0 }
00190       };
00191       if (FAILED(gfx.Dev()->CreateInputLayout(il, 2, vsb->GetBufferPointer(), vsb->GetBufferSize(),
      layout_.GetAddressOf()))) {
00192           return false;
00193       }
00194
00195       // VS 定数バッファ
00196       D3D11_BUFFER_DESC cbd{};
00197       cbd.ByteWidth = sizeof(VSConstants);
00198       cbd.Usage = D3D11_USAGE_DEFAULT;
00199       cbd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
00200       if (FAILED(gfx.Dev()->CreateBuffer(&cbd, nullptr, cb_.GetAddressOf()))) {
00201           return false;
00202       }
00203
00204       // PS 定数バッファ
00205       cbd.ByteWidth = sizeof(PSConstants);
00206       if (FAILED(gfx.Dev()->CreateBuffer(&cbd, nullptr, psCb_.GetAddressOf()))) {
00207           return false;
00208       }
00209
00210       // サンプラーステート
00211       D3D11_SAMPLER_DESC sampDesc{};
00212       sampDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
00213       sampDesc.AddressU = D3D11_TEXTURE_ADDRESS_WRAP;
00214       sampDesc.AddressV = D3D11_TEXTURE_ADDRESS_WRAP;
00215       sampDesc.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;
00216       sampDesc.MaxAnisotropy = 1;
00217       sampDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;
00218       sampDesc.MinLOD = 0;
```

```
00219         sampDesc.MaxLOD = D3D11_FLOAT32_MAX;
00220         if (FAILED(gfx.Dev()->CreateSamplerState(&sampDesc, &samplerState_))) {
00221             return false;
00222         }
00223 
00224         // ラスタライザステート
00225         D3D11_RASTERIZER_DESC rsd{};
00226         rsd.FillMode = D3D11_FILL_SOLID;
00227         rsd.CullMode = D3D11_CULL_BACK;
00228         rsd.FrontCounterClockwise = FALSE;
00229         rsd.DepthClipEnable = TRUE;
00230         if (FAILED(gfx.Dev()->CreateRasterizerState(&rsd, rasterState_.GetAddressOf()))) {
00231             return false;
00232         }
00233 
00234         // ジオメトリ（UV 座標付きキューブ）
00235         struct V { DirectX::XMFLOAT3 pos; DirectX::XMFLOAT2 tex; };
00236         const float c = 0.5f;
00237         V verts[] = {
00238             // 背面
00239             {{-c,-c,-c}, {0,1}}, {{-c,+c,-c}, {0,0}}, {{+c,+c,-c}, {1,0}}, {{+c,-c,-c}, {1,1}},
00240             // 前面
00241             {{-c,-c,+c}, {1,1}}, {{-c,+c,+c}, {1,0}}, {{+c,+c,+c}, {0,0}}, {{+c,-c,+c}, {0,1}},
00242         };
00243         uint16_t idx[] = {
00244             0,1,2, 0,2,3,  // 背面
00245             4,6,5, 4,7,6,  // 前面
00246             4,5,1, 4,1,0,  // 左
00247             3,2,6, 3,6,7,  // 右
00248             1,5,6, 1,6,2,  // 上
00249             4,0,3, 4,3,7   // 下
00250         };
00251         indexCount_ = (UINT)(sizeof(idx) / sizeof(idx[0]));
00252 
00253         D3D11_BUFFER_DESC vbd{};
00254         vbd.ByteWidth = (UINT)sizeof(verts);
00255         vbd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
00256         vbd.Usage = D3D11_USAGE_IMMUTABLE;
00257         D3D11_SUBRESOURCE_DATA vinit{ verts, 0, 0 };
00258         if (FAILED(gfx.Dev()->CreateBuffer(&vbd, &vinit, vb_.GetAddressOf()))) {
00259             return false;
00260         }
00261 
00262         D3D11_BUFFER_DESC ibd{};
00263         ibd.ByteWidth = (UINT)sizeof(idx);
00264         ibd.BindFlags = D3D11_BIND_INDEX_BUFFER;
00265         ibd.Usage = D3D11_USAGE_IMMUTABLE;
00266         D3D11_SUBRESOURCE_DATA iinit{ idx, 0, 0 };
00267         if (FAILED(gfx.Dev()->CreateBuffer(&ibd, &iinit, ib_.GetAddressOf()))) {
00268             return false;
00269         }
00270 
00271         return true;
00272     }
00273 
00283     void Render(GfxDevice& gfx, World& w, const Camera& cam) {
00284         gfx.Ctx()->IASetInputLayout(layout_.Get());
00285         gfx.Ctx()->VSSetShader(vs_.Get(), nullptr, 0);
00286         gfx.Ctx()->PSSetShader(ps_.Get(), nullptr, 0);
00287         gfx.Ctx()->VSSetConstantBuffers(0, 1, cb_.GetAddressOf());
00288         gfx.Ctx()->PSSetConstantBuffers(0, 1, psCb_.GetAddressOf());
00289         gfx.Ctx()->PSSetSamplers(0, 1, samplerState_.GetAddressOf());
00290         gfx.Ctx()->RSSetState(rasterState_.Get());
00291 
00292         UINT stride = sizeof(DirectX::XMFLOAT3) + sizeof(DirectX::XMFLOAT2);
00293         UINT offset = 0;
00294         gfx.Ctx()->IASetVertexBuffers(0, 1, vb_.GetAddressOf(), &stride, &offset);
00295         gfx.Ctx()->IASetIndexBuffer(ib_.Get(), DXGI_FORMAT_R16_UINT, 0);
00296         gfx.Ctx()->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
00297 
00298         w.ForEach<MeshRenderer>([&](Entity e, MeshRenderer& mr) {
00299             auto* t = w.TryGet<Transform>(e);
00300             if (!t) return;
00301 
00302             // ワールド行列
00303             DirectX::XMMATRIX S = DirectX::XMMatrixScaling(t->scale.x, t->scale.y, t->scale.z);
00304             DirectX::XMMATRIX R = DirectX::XMMatrixRotationRollPitchYaw(
00305                 DirectX::XMConvertToRadians(t->rotation.x),
00306                 DirectX::XMConvertToRadians(t->rotation.y),
00307                 DirectX::XMConvertToRadians(t->rotation.z));
00308             DirectX::XMMATRIX T = DirectX::XMMatrixTranslation(t->position.x, t->position.y, t->position.z);
00309             DirectX::XMMATRIX W = S * R * T;
00310 
00311             // VS 定数バッファ
00312             VSConstants vsCbuf;
00313             vsCbuf.WVP = DirectX::XMMatrixTranspose(W * cam.View * cam.Proj);
00314             vsCbuf.uvTransform = DirectX::XMFLOAT4{ mr.uvOffset.x, mr.uvOffset.y, mr.uvScale.x, mr.uvScale.y };
```

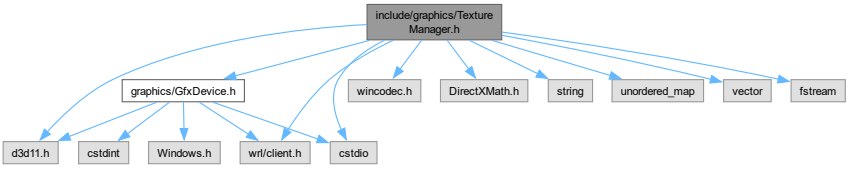
```
00315             gfx.Ctx()->UpdateSubresource(cb_.Get(), 0, nullptr, &vsCbuf, 0, 0);
00316
00317             // PS 定数バッファ
00318             PSConstants psCbuf;
00319             psCbuf.color = DirectX::XMFLOAT4{ mr.color.x, mr.color.y, mr.color.z, 1.0f };
00320             psCbuf.useTexture = (mr.texture != TextureManager::INVALID_TEXTURE) ? 1.0f : 0.0f;
00321             gfx.Ctx()->UpdateSubresource(psCb_.Get(), 0, nullptr, &psCbuf, 0, 0);
00322
00323             // テクスチャ設定
00324             if (mr.texture != TextureManager::INVALID_TEXTURE && texManager_) {
00325                 ID3D11ShaderResourceView* srv = texManager_->GetSRV(mr.texture);
00326                 if (srv) {
00327                     gfx.Ctx()->PSSetShaderResources(0, 1, &srv);
00328                 }
00329             }
00330
00331             gfx.Ctx()->DrawIndexed(indexCount_, 0, 0);
00332         });
00333     }
00334 };
```

7.25 include/graphics/TextureManager.h ファイル

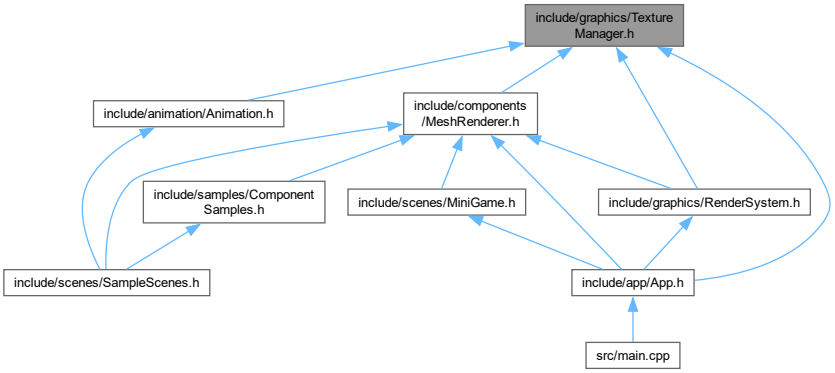
テクスチャ管理システム

```
#include "graphics/GfxDevice.h"
#include <d3d11.h>
#include <wrl/client.h>
#include <wincodec.h>
#include <DirectXMath.h>
#include <string>
#include <unordered_map>
#include <vector>
#include <fstream>
#include <cstdio>
```

TextureManager.h の依存先関係図:



被依存関係図:



クラス

- class TextureManager
  テクスチャ管理システム

7.25.1 詳解

テクスチャ管理システム

著者

山内陽

日付

2024

バージョン

5.0

画像ファイルの読み込み、テクスチャの作成・管理を行うシステムです。WIC (Windows Imaging Component) を使用して様々な画像フォーマットに対応しています。

7.26 TextureManager.h

[詳解]

```
00001
00012 #pragma once
00013 #include "graphics/GfxDevice.h"
00014 #include <d3d11.h>
00015 #include <wrl/client.h>
00016 #include <wincodec.h>
00017 #include <DirectXMath.h>
00018 #include <string>
00019 #include <unordered_map>
00020 #include <vector>
00021 #include <fstream>
00022 #include <cstdio>
00023
00024 #pragma comment(lib, "windowscodecs.lib")
00025
00062 class TextureManager {
00063 public:
00068     using TextureHandle = uint32_t;
00069
00074     static constexpr TextureHandle INVALID_TEXTURE = 0;
00075
00081     bool Init(GfxDevice& gfx) {
00082         gfx_ = &gfx;
00083
00084         // デフォルトの白テクスチャを作成
00085         uint32_t whitePixel = 0xFFFFFFFF;
00086         defaultWhiteTexture_ = CreateTextureFromMemory(
00087             reinterpret_cast<const uint8_t*>(&whitePixel),
00088             1, 1, 4
00089         );
00090
00091         return defaultWhiteTexture_ != INVALID_TEXTURE;
00092     }
00093
00111     TextureHandle LoadFromFile(const char* filepath) {
00112         // WIC を使用して画像を読み込む
```

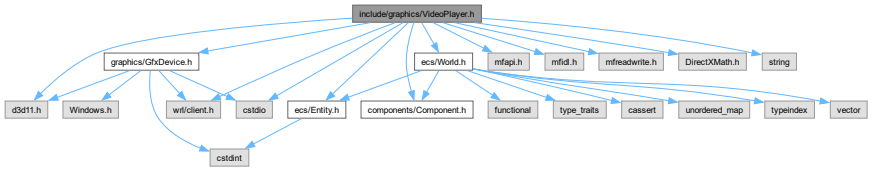
```
00113       Microsoft::WRL::ComPtr<IWICImagingFactory> wicFactory;
00114       HRESULT hr = CoCreateInstance(
00115           CLSID_WICImagingFactory,
00116           nullptr,
00117           CLSCTX_INPROC_SERVER,
00118           IID_PPV_ARGS(&wicFactory)
00119       );
00120
00121       if (FAILED(hr)) {
00122           char msg[512];
00123           sprintf_s(msg, "Failed to create WIC factory for: %s", filepath);
00124           MessageBoxA(nullptr, msg, "Texture Load Error", MB_OK | MB_ICONERROR);
00125           return INVALID_TEXTURE;
00126       }
00127
00128       // ワイド文字列に変換
00129       wchar_t wpath[MAX_PATH];
00130       MultiByteToWideChar(CP_ACP, 0, filepath, -1, wpath, MAX_PATH);
00131
00132       // デコーダーを作成
00133       Microsoft::WRL::ComPtr<IWICBitmapDecoder> decoder;
00134       hr = wicFactory->CreateDecoderFromFilename(
00135           wpath,
00136           nullptr,
00137           GENERIC_READ,
00138           WICDecodeMetadataCacheOnDemand,
00139           &decoder
00140       );
00141
00142       if (FAILED(hr)) {
00143           char msg[512];
00144           sprintf_s(msg, "Failed to load image file: %s", filepath);
00145           MessageBoxA(nullptr, msg, "Texture Load Error", MB_OK | MB_ICONERROR);
00146           return INVALID_TEXTURE;
00147       }
00148
00149       // フレームを取得
00150       Microsoft::WRL::ComPtr<IWICBitmapFrameDecode> frame;
00151       hr = decoder->GetFrame(0, &frame);
00152       if (FAILED(hr)) return INVALID_TEXTURE;
00153
00154       // RGBA32 に変換
00155       Microsoft::WRL::ComPtr<IWICFormatConverter> converter;
00156       hr = wicFactory->CreateFormatConverter(&converter);
00157       if (FAILED(hr)) return INVALID_TEXTURE;
00158
00159       hr = converter->Initialize(
00160           frame.Get(),
00161           GUID_WICPixelFormat32bppRGBA,
00162           WICBitmapDitherTypeNone,
00163           nullptr,
00164           0.0,
00165           WICBitmapPaletteTypeCustom
00166       );
00167       if (FAILED(hr)) return INVALID_TEXTURE;
00168
00169       // サイズを取得
00170       UINT width, height;
00171       hr = converter->GetSize(&width, &height);
00172       if (FAILED(hr)) return INVALID_TEXTURE;
00173
00174       // ピクセルデータを取得
00175       std::vector<uint8_t> pixels(width * height * 4);
00176       hr = converter->CopyPixels(
00177           nullptr,
00178           width * 4,
00179           static_cast<UINT>(pixels.size()),
00180           pixels.data()
00181       );
00182       if (FAILED(hr)) return INVALID_TEXTURE;
00183
00184       return CreateTextureFromMemory(pixels.data(), width, height, 4);
00185   }
00186
00199   TextureHandle CreateTextureFromMemory(const uint8_t* data, uint32_t width, uint32_t height, uint32_t channels) {
00200       D3D11_TEXTURE2D_DESC texDesc{};
00201       texDesc.Width = width;
00202       texDesc.Height = height;
00203       texDesc.MipLevels = 1;
00204       texDesc.ArraySize = 1;
00205       texDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
00206       texDesc.SampleDesc.Count = 1;
00207       texDesc.Usage = D3D11_USAGE_DEFAULT;
00208       texDesc.BindFlags = D3D11_BIND_SHADER_RESOURCE;
00209
00210       D3D11_SUBRESOURCE_DATA initData{};
00211       initData.pSysMem = data;
```

```
00212         initData.SysMemPitch = width * channels;
00213
00214         Microsoft::WRL::ComPtr<ID3D11Texture2D> texture;
00215         HRESULT hr = gfx_->Dev()->CreateTexture2D(&texDesc, &initData, &texture);
00216         if (FAILED(hr)) {
00217             MessageBoxA(nullptr, "Failed to create texture2D", "Texture Error", MB_OK | MB_ICONERROR);
00218             return INVALID_TEXTURE;
00219         }
00220
00221         // シェーダーリソースビューを作成
00222         D3D11_SHADER_RESOURCE_VIEW_DESC srvDesc{};
00223         srvDesc.Format = texDesc.Format;
00224         srvDesc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
00225         srvDesc.Texture2D.MipLevels = 1;
00226
00227         Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> srv;
00228         hr = gfx_->Dev()->CreateShaderResourceView(texture.Get(), &srvDesc, &srv);
00229         if (FAILED(hr)) {
00230             MessageBoxA(nullptr, "Failed to create SRV", "Texture Error", MB_OK | MB_ICONERROR);
00231             return INVALID_TEXTURE;
00232         }
00233
00234         // テクスチャを登録
00235         TextureHandle handle = nextHandle_++;
00236         TextureData texData;
00237         texData.texture = texture;
00238         texData.srv = srv;
00239         texData.width = width;
00240         texData.height = height;
00241         textures_[handle] = texData;
00242
00243         return handle;
00244     }
00245
00255     ID3D11ShaderResourceView* GetSRV(TextureHandle handle) const {
00256         if (handle == INVALID_TEXTURE) return nullptr;
00257         auto it = textures_.find(handle);
00258         if (it == textures_.end()) return nullptr;
00259         return it->second.srv.Get();
00260     }
00261
00270     TextureHandle GetDefaultWhite() const { return defaultWhiteTexture_; }
00271
00280     void Release(TextureHandle handle) {
00281         textures_.erase(handle);
00282     }
00283
00287     ~TextureManager() {
00288         textures_.clear();
00289     }
00290
00291 private:
00296     struct TextureData {
00297         Microsoft::WRL::ComPtr<ID3D11Texture2D> texture;
00298         Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> srv;
00299         uint32_t width;
00300         uint32_t height;
00301     };
00302
00303     GfxDevice* gfx_ = nullptr;
00304     std::unordered_map<TextureHandle, TextureData> textures_;
00305     TextureHandle nextHandle_ = 1;
00306     TextureHandle defaultWhiteTexture_ = INVALID_TEXTURE;
00307 };
```

7.27 include/graphics/VideoPlayer.h ファイル

```
#include "graphics/GfxDevice.h"
#include "components/Component.h"
#include "ecs/Entity.h"
#include "ecs/World.h"
#include <d3d11.h>
#include <mfapi.h>
#include <mfidl.h>
#include <mfreadwrite.h>
#include <wrl/client.h>
#include <DirectXMath.h>
```

#include <string>
#include <cstdio>
VideoPlayer.h の依存先関係図:



クラス

- class VideoPlayer
- struct VideoPlayback

7.28 VideoPlayer.h

[詳解]

```
00001 #pragma once
00002 #include "graphics/GfxDevice.h"
00003 #include "components/Component.h"
00004 #include "ecs/Entity.h"
00005 #include "ecs/World.h"
00006 #include <d3d11.h>
00007 #include <mfapi.h>
00008 #include <mfidl.h>
00009 #include <mfreadwrite.h>
00010 #include <wrl/client.h>
00011 #include <DirectXMath.h>
00012 #include <string>
00013 #include <cstdio>
00014
00015 #pragma comment(lib, "mf.lib")
00016 #pragma comment(lib, "mfplat.lib")
00017 #pragma comment(lib, "mfreadwrite.lib")
00018 #pragma comment(lib, "mfuuid.lib")
00019
00020 // ============================================================
00021 // VideoPlayer - ♦♦♦♦Đ♦♦V♦X♦e♦♦
00022 // ============================================================
00023 class VideoPlayer {
00024 public:
00025     bool Init() {
00026         // Media Foundation♦♦♦♦♦♦
00027         HRESULT hr = MFStartup(MF_VERSION);
00028         if (FAILED(hr)) {
00029             MessageBoxA(nullptr, "Failed to initialize Media Foundation", "Video Error", MB_OK | MB_ICONERROR);
00030             return false;
00031         }
00032         return true;
00033     }
00034
00035     ~VideoPlayer() {
00036         if (reader_) reader_.Reset();
00037         MFShutdown();
00038     }
00039
00040     // ♦♦♦♦t♦@♦C♦♦♦♦♦J♦♦
00041     bool Open(GfxDevice& gfx, const char* filepath) {
00042         gfx_ = &gfx;
00043
00044         // ♦♦♦C♦h♦♦♦♦♦♦ ♦
00045         wchar_t wpath[MAX_PATH];
00046         MultiByteToWideChar(CP_ACP, 0, filepath, -1, wpath, MAX_PATH);
00047
00048         // ♦\♦[♦X♦♦♦[♦_♦[♦♦♦□
00049         Microsoft::WRL::ComPtr<IMFAttributes> attributes;
00050         HRESULT hr = MFCreateAttributes(&attributes, 1);
```

```
00051         if (FAILED(hr)) return false;
00052
00053         hr = attributes->SetUINT32(MF_SOURCE_READER_ENABLE_VIDEO_PROCESSING, TRUE);
00054         if (FAILED(hr)) return false;
00055
00056         hr = MFCreateSourceReaderFromURL(wpath, attributes.Get(), &reader_);
00057         if (FAILED(hr)) {
00058             char msg[512];
00059             sprintf_s(msg, "Failed to open video file: %s", filepath);
00060             MessageBoxA(nullptr, msg, "Video Error", MB_OK | MB_ICONERROR);
00061             return false;
00062         }
00063
00064         // ♦r♦f♦I♦X♦g♦♦♦[♦♦♦♦I♦♦
00065         hr = reader_->SetStreamSelection((DWORD)MF_SOURCE_READER_FIRST_VIDEO_STREAM, TRUE);
00066         if (FAILED(hr)) return false;
00067
00068         // ♦o♦♦♦f♦B♦A♦^♦C♦v♦♦ ♦iRGB32♦j
00069         Microsoft::WRL::ComPtr<IMFMediaType> mediaType;
00070         hr = MFCreateMediaType(&mediaType);
00071         if (FAILED(hr)) return false;
00072
00073         hr = mediaType->SetGUID(MF_MT_MAJOR_TYPE, MFMediaType_Video);
00074         if (FAILED(hr)) return false;
00075
00076         hr = mediaType->SetGUID(MF_MT_SUBTYPE, MFVideoFormat_RGB32);
00077         if (FAILED(hr)) return false;
00078
00079         hr = reader_->SetCurrentMediaType((DWORD)MF_SOURCE_READER_FIRST_VIDEO_STREAM, nullptr,
    mediaType.Get());
00080         if (FAILED(hr)) return false;
00081
00082         // ♦♦♦♦T♦C♦Y♦f♦♦♦♦♦♦擾
00083         Microsoft::WRL::ComPtr<IMFMediaType> currentType;
00084         hr = reader_->GetCurrentMediaType((DWORD)MF_SOURCE_READER_FIRST_VIDEO_STREAM,
    &currentType);
00085         if (FAILED(hr)) return false;
00086
00087         UINT32 w, h;
00088         hr = MFGetAttributeSize(currentType.Get(), MF_MT_FRAME_SIZE, &w, &h);
00089         if (FAILED(hr)) return false;
00090
00091         width_ = w;
00092         height_ = h;
00093
00094         // ♦♦♦I♦e♦N♦X♦‘♦♦♦♦♦□
00095         if (!createVideoTexture()) return false;
00096
00097         isOpen_ = true;
00098         return true;
00099     }
00100
00101     // ♦t♦♦♦[♦♦♦♦♦X♦V
00102     bool Update(float dt) {
00103         if (!isOpen_ || !isPlaying_) return false;
00104
00105         currentTime_ += dt;
00106
00107         // ♦t♦♦♦[♦♦♦♦ ♦♦♦
00108         DWORD streamFlags = 0;
00109         LONGLONG timestamp = 0;
00110         Microsoft::WRL::ComPtr<IMFSample> sample;
00111
00112         HRESULT hr = reader_->ReadSample(
00113             (DWORD)MF_SOURCE_READER_FIRST_VIDEO_STREAM,
00114             0,
00115             nullptr,
00116             &streamFlags,
00117             &timestamp,
00118             &sample
00119         );
00120
00121         if (FAILED(hr)) return false;
00122
00123         // ♦X♦g♦♦♦[♦♦♦I♦[
00124         if (streamFlags & MF_SOURCE_READERF_ENDOFSTREAM) {
00125             if (loop_) {
00126                 // ♦♦♦[♦v♦Đ♦
00127                 PROPVARIANT var{};
00128                 var.vt = VT_I8;
00129                 var.hVal.QuadPart = 0;
00130                 reader_->SetCurrentPosition(GUID_NULL, var);
00131                 PropVariantClear(&var);
00132                 return true;
00133             } else {
00134                 isPlaying_ = false;
00135                 return false;
```

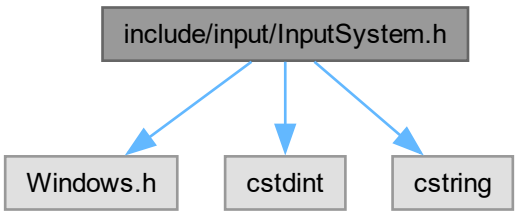
```
00136             }
00137         }
00138
00139         if (!sample) return true;
00140
00141         // ♦T♦♦♦v♦♦♦♦♦♦o♦b♦t♦@♦♦♦擾
00142         Microsoft::WRL::ComPtr<IMFMediaBuffer> buffer;
00143         hr = sample->ConvertToContiguousBuffer(&buffer);
00144         if (FAILED(hr)) return false;
00145
00146         BYTE* data = nullptr;
00147         DWORD length = 0;
00148         hr = buffer->Lock(&data, nullptr, &length);
00149         if (FAILED(hr)) return false;
00150
00151         // ♦e♦N♦X♦‘♦♦♦♦X♦V
00152         D3D11_MAPPED_SUBRESOURCE mapped;
00153         hr = gfx_->Ctx()->Map(videoTexture_.Get(), 0, D3D11_MAP_WRITE_DISCARD, 0, &mapped);
00154         if (SUCCEEDED(hr)) {
00155             // ♦f♦[♦^♦♦♦R♦s♦[
00156             uint8_t* dest = static_cast<uint8_t*>(mapped.pData);
00157             uint8_t* src = data;
00158
00159             for (UINT y = 0; y < height_; ++y) {
00160                 memcpy(dest, src, width_ * 4);
00161                 dest += mapped.RowPitch;
00162                 src += width_ * 4;
00163             }
00164
00165             gfx_->Ctx()->Unmap(videoTexture_.Get(), 0);
00166         }
00167
00168         buffer->Unlock();
00169         return true;
00170     }
00171
00172     // ♦Đ♦♦E♦♦~
00173     void Play() { isPlaying_ = true; }
00174     void Stop() { isPlaying_ = false; }
00175     void SetLoop(bool loop) { loop_ = loop; }
00176
00177     // ♦e♦N♦X♦‘♦♦♦♦擾
00178     ID3D11ShaderResourceView* GetSRV() const { return videoSRV_.Get(); }
00179
00180     bool IsPlaying() const { return isPlaying_; }
00181     UINT GetWidth() const { return width_; }
00182     UINT GetHeight() const { return height_; }
00183
00184 private:
00185     bool createVideoTexture() {
00186         D3D11_TEXTURE2D_DESC texDesc{};
00187         texDesc.Width = width_;
00188         texDesc.Height = height_;
00189         texDesc.MipLevels = 1;
00190         texDesc.ArraySize = 1;
00191         texDesc.Format = DXGI_FORMAT_B8G8R8A8_UNORM;
00192         texDesc.SampleDesc.Count = 1;
00193         texDesc.Usage = D3D11_USAGE_DYNAMIC;
00194         texDesc.BindFlags = D3D11_BIND_SHADER_RESOURCE;
00195         texDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
00196
00197         HRESULT hr = gfx_->Dev()->CreateTexture2D(&texDesc, nullptr, &videoTexture_);
00198         if (FAILED(hr)) {
00199             MessageBoxA(nullptr, "Failed to create video texture", "Video Error", MB_OK | MB_ICONERROR);
00200             return false;
00201         }
00202
00203         D3D11_SHADER_RESOURCE_VIEW_DESC srvDesc{};
00204         srvDesc.Format = texDesc.Format;
00205         srvDesc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
00206         srvDesc.Texture2D.MipLevels = 1;
00207
00208         hr = gfx_->Dev()->CreateShaderResourceView(videoTexture_.Get(), &srvDesc, &videoSRV_);
00209         if (FAILED(hr)) {
00210             MessageBoxA(nullptr, "Failed to create video SRV", "Video Error", MB_OK | MB_ICONERROR);
00211             return false;
00212         }
00213
00214         return true;
00215     }
00216
00217     GfxDevice* gfx_ = nullptr;
00218     Microsoft::WRL::ComPtr<IMFSourceReader> reader_;
00219     Microsoft::WRL::ComPtr<ID3D11Texture2D> videoTexture_;
00220     Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> videoSRV_;
00221
00222     UINT width_ = 0;
```

```
00223     UINT height_ = 0;
00224     bool isOpen_ = false;
00225     bool isPlaying_ = false;
00226     bool loop_ = false;
00227     float currentTime_ = 0.0f;
00228 };
00229
00230 // ============================================================
00231 // VideoPlayback - ♦♦♦♦Đ♦♦R♦♦♦|♦[♦]♦♦♦g
00232 // ============================================================
00233 struct VideoPlayback : Behaviour {
00234     VideoPlayer* player = nullptr; // VideoPlayer♦'|♦C♦♦♦^
00235     bool autoPlay = true;
00236
00237     void OnStart(World& w, Entity self) override {
00238         if (autoPlay && player) {
00239             player->Play();
00240         }
00241     }
00242
00243     void OnUpdate(World& w, Entity self, float dt) override {
00244         if (player) {
00245             player->Update(dt);
00246         }
00247     }
00248 };
```

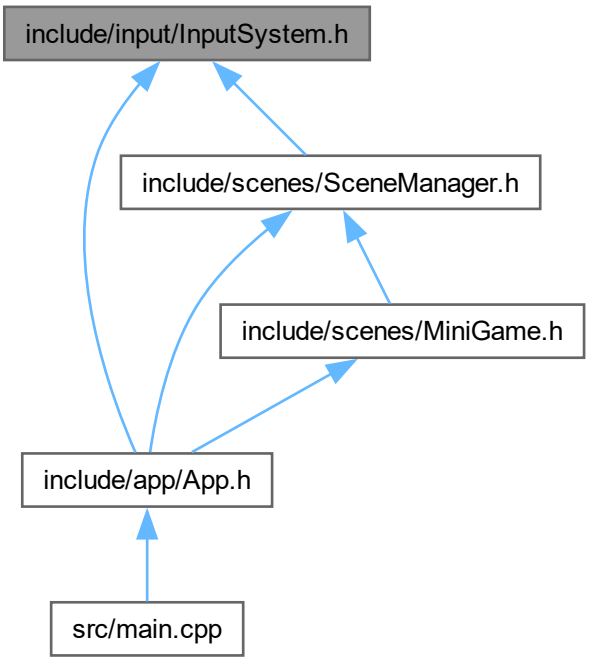
7.29 include/input/InputSystem.h ファイル

キーボード・マウス入力管理システム

#include <Windows.h>
#include <cstdint>
#include <cstring>
InputSystem.h の依存先関係図:



被依存関係図:



クラス

- class InputSystem
  キーボード・マウス入力を管理するクラス

マクロ定義

- #define WIN32_LEAN_AND_MEAN
- #define NOMINMAX

関数

- InputSystem & GetInput ()
  グローバルな入力システムインスタンスを取得

7.29.1 詳解

キーボード・マウス入力管理システム

著者

山内陽

日付

2024

バージョン

5.0

このファイルはキーボードとマウスの入力を管理するシステムを提供します。

7.29.2 マクロ定義詳解

7.29.2.1 NOMINMAX

#define NOMINMAX

7.29.2.2 WIN32_LEAN_AND_MEAN

#define WIN32_LEAN_AND_MEAN

7.29.3 関数詳解

7.29.3.1 GetInput()

InputSystem & GetInput () [inline]

グローバルな入力システムインスタンスを取得

戻り値

InputSystem& シングルトンインスタンス

著者

山内陽

7.30 InputSystem.h

[詳解]

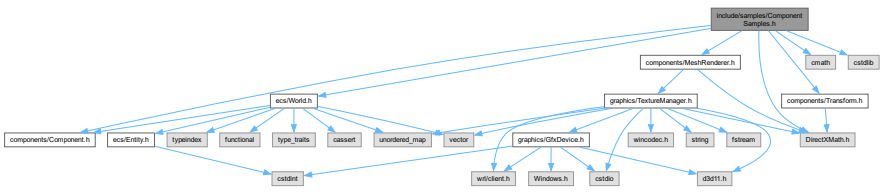
```
00001 #pragma once
00002 #define WIN32_LEAN_AND_MEAN
00003 #define NOMINMAX
00004 #include <Windows.h>
00005 #include <cstdint>
00006 #include <cstring>
00007
00018
00043 class InputSystem {
00044 public:
00049     enum class KeyState : uint8_t {
00050         None = 0,
00051         Down = 1,
00052         Pressed = 2,
00053         Up = 3
00054     };
00055
00060     enum MouseButton {
00061         Left = 0,
00062         Right = 1,
00063         Middle = 2
00064     };
00065
00069     void Init() {
00070         memset(keyStates_, 0, sizeof(keyStates_));
00071         memset(prevKeyStates_, 0, sizeof(prevKeyStates_));
00072         memset(mouseStates_, 0, sizeof(mouseStates_));
00073         memset(prevMouseStates_, 0, sizeof(prevMouseStates_));
00074         mouseX_ = mouseY_ = 0;
00075         mouseDeltaX_ = mouseDeltaY_ = 0;
00076         mouseWheel_ = 0;
00077     }
00078
00082     void Update() {
00083         memcpy(prevKeyStates_, keyStates_, sizeof(keyStates_));
00084         memcpy(prevMouseStates_, mouseStates_, sizeof(mouseStates_));
00085
00086         for (int i = 0; i < 256; ++i) {
00087             bool current = (GetAsyncKeyState(i) & 0x8000) != 0;
00088             bool prev = prevKeyStates_[i];
00089
00090             if (current && !prev) {
00091                 keyStates_[i] = static_cast<uint8_t>(KeyState::Down);
00092             } else if (current && prev) {
00093                 keyStates_[i] = static_cast<uint8_t>(KeyState::Pressed);
00094             } else if (!current && prev) {
00095                 keyStates_[i] = static_cast<uint8_t>(KeyState::Up);
00096             } else {
00097                 keyStates_[i] = static_cast<uint8_t>(KeyState::None);
00098             }
00099         }
00100
00101         POINT pt;
00102         if (GetCursorPos(&pt)) {
00103             int newX = pt.x;
00104             int newY = pt.y;
00105             mouseDeltaX_ = newX - mouseX_;
00106             mouseDeltaY_ = newY - mouseY_;
00107             mouseX_ = newX;
00108             mouseY_ = newY;
00109         }
00110
00111         mouseWheel_ = 0;
00112     }
00113
00119     bool GetKey(int vkCode) const {
00120         if (vkCode < 0 || vkCode >= 256) return false;
00121         KeyState state = static_cast<KeyState>(keyStates_[vkCode]);
00122         return state == KeyState::Pressed || state == KeyState::Down;
00123     }
00124
00130     bool GetKeyDown(int vkCode) const {
00131         if (vkCode < 0 || vkCode >= 256) return false;
00132         return static_cast<KeyState>(keyStates_[vkCode]) == KeyState::Down;
00133     }
00134
00140     bool GetKeyUp(int vkCode) const {
00141         if (vkCode < 0 || vkCode >= 256) return false;
00142         return static_cast<KeyState>(keyStates_[vkCode]) == KeyState::Up;
00143     }
00144
00150     bool GetMouseButton(MouseButton button) const {
```

```
00151         int vk = VK_LBUTTON;
00152         if (button == Right) vk = VK_RBUTTON;
00153         else if (button == Middle) vk = VK_MBUTTON;
00154         return GetKey(vk);
00155     }
00156
00162     bool GetMouseButtonDown(MouseButton button) const {
00163         int vk = VK_LBUTTON;
00164         if (button == Right) vk = VK_RBUTTON;
00165         else if (button == Middle) vk = VK_MBUTTON;
00166         return GetKeyDown(vk);
00167     }
00168
00174     bool GetMouseButtonUp(MouseButton button) const {
00175         int vk = VK_LBUTTON;
00176         if (button == Right) vk = VK_RBUTTON;
00177         else if (button == Middle) vk = VK_MBUTTON;
00178         return GetKeyUp(vk);
00179     }
00180
00185     int GetMouseX() const { return mouseX_; }
00186
00191     int GetMouseY() const { return mouseY_; }
00192
00197     int GetMouseDeltaX() const { return mouseDeltaX_; }
00198
00203     int GetMouseDeltaY() const { return mouseDeltaY_; }
00204
00209     int GetMouseWheel() const { return mouseWheel_; }
00210
00215     void OnMouseWheel(int delta) {
00216         mouseWheel_ = delta / 120;
00217     }
00218
00219 private:
00220     uint8_t keyStates_[256];
00221     uint8_t prevKeyStates_[256];
00222     uint8_t mouseStates_[3];
00223     uint8_t prevMouseStates_[3];
00224
00225     int mouseX_;
00226     int mouseY_;
00227     int mouseDeltaX_;
00228     int mouseDeltaY_;
00229     int mouseWheel_;
00230 };
00231
00238 inline InputSystem& GetInput() {
00239     static InputSystem instance;
00240     return instance;
00241 }
```

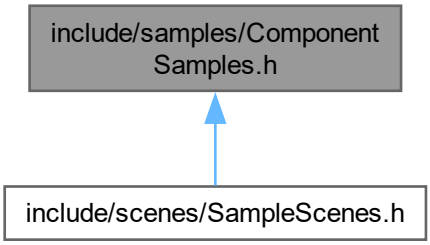
7.31 include/samples/ComponentSamples.h ファイル

学習用コンポーネント集

#include ”components/Component.h”
#include ”ecs/World.h”
#include ”components/Transform.h”
#include ”components/MeshRenderer.h”
#include <DirectXMath.h>
#include <cmath>
#include <cstdlib>
ComponentSamples.h の依存先関係図:



被依存関係図:



クラス

- struct Health
  データ型のコンポーネント
- struct Velocity
  速度コンポーネント
- struct PlayerTag
  タグコンポーネント（データなし）プレイヤータグ
- struct EnemyTag
  敵タグ
- struct BulletTag
  弾タグ
- struct Bouncer
  シンプルなBehaviour
- struct MoveForward
  前に進むBehaviour
- struct PulseScale
  拡大縮小（パルス）Behaviour
- struct ColorCycle
  色を変化（サイクル）Behaviour
- struct DestroyOnDeath
  複雑なBehaviour
- struct RandomWalk
  ランダムに動き回るBehaviour
- struct LifeTime
  時間経過で削除するBehaviour

関数

- DEFINE_DATA_COMPONENT (Score, int points=0;void AddPoints(int p) { points+=p;} void Reset() { points=0;})
  スコアコンポーネント（マクロ版）
- DEFINE_DATA_COMPONENT (Name, const char *name="Unnamed";)
  名前コンポーネント（マクロ版）

- DEFINE_BEHAVIOUR (SpinAndColor, float rotSpeed=90.0f;float colorSpeed=1.0f;float time=0.0f;, time+=dt ∗colorSpeed;auto ∗t=w.TryGet< Transform >(self);if(t) { t->rotation.↩ y+=rotSpeed ∗dt;} auto ∗mr=w.TryGet< MeshRenderer >(self);if(mr) { float hue=fmodf(time, 1.0f);mr->color.x=sinf(hue ∗6.28f) ∗0.5f+0.5f;mr->color.y=cosf(hue ∗6.28f) ∗0.5f+0.5f;mr->color.z=0.5f;})

  マクロを使った簡潔な定義
- DEFINE_BEHAVIOUR (CircularMotion, float radius=3.0f;float speed=1.0f;float angle=0.0f;float centerY=0.0f;, angle+=speed ∗dt;auto ∗t=w.TryGet< Transform >(self);if(t) { t->position.↩ x=cosf(angle) ∗radius;t->position.z=sinf(angle) ∗radius;t->position.y=centerY;})

  円運動を行う（マクロ版）

7.31.1 詳解

学習用コンポーネント集

著者

山内陽

日付

2024

バージョン

4.0

コピペして使える実用的なコンポーネントサンプル集です。学習方針：コードを読む → 理解する → 改造する

7.31.2 関数詳解

7.31.2.1 DEFINE_BEHAVIOUR() [1/2]

```
DEFINE_BEHAVIOUR (
        CircularMotion ,
        float radius = 3.0f;float speed=1.0f;float angle=0.0f;float centerY=0.0f;,
        angle+  = speed ∗dt;auto ∗t=w.TryGet< Transform >(self);if(t) { t->position.x=cosf(angle) ∗radius;t->position.z=sinf(angle) ∗radius;t->position.y=centerY;})
```

円運動を行う（マクロ版）

引数

radius	円の半径回転速度現在の角度中心のY 座標

7.31.2.2 DEFINE_BEHAVIOUR() [2/2]

```
DEFINE_BEHAVIOUR (
        SpinAndColor ,
        float rotSpeed = 90.0f;float colorSpeed=1.0f;float time=0.0f;,
        time+ = dt *colorSpeed;auto *t=w.TryGet< Transform >(self);if(t) { t->rotation.y+=rotSpeed *dt;} auto *mr=w.TryGet< Me
>color.x=sinf(hue *6.28f) *0.5f+0.5f;mr->color.y=cosf(hue *6.28f) *0.5f+0.5f;mr->color.z=0.5f;})
```

マクロを使った簡潔な定義

DEFINE_BEHAVIOUR マクロで短く書ける学習ポイント：ボイラープレートの削減

回転しながら色を変える（マクロ版）

引数

rotSpeed	回転速度色変化速度経過時間

7.31.2.3 DEFINE_DATA_COMPONENT() [1/2]

```
DEFINE_DATA_COMPONENT (
        Name ,
        const char * name = "Unnamed";)
```

名前コンポーネント（マクロ版）

引数

name	エンティティ名

7.31.2.4 DEFINE_DATA_COMPONENT() [2/2]

```
DEFINE_DATA_COMPONENT (
        Score ,
        int points = 0;void AddPoints(int p) { points+=p;} void Reset() { points=0;})
```

スコアコンポーネント（マクロ版）

引数

	points	獲得ポ
in	p	

スコアをリセット

7.32 ComponentSamples.h

[詳解]

```
00001
00012 #pragma once
00013
00014 #include "components/Component.h"
00015 #include "ecs/World.h"
00016 #include "components/Transform.h"
00017 #include "components/MeshRenderer.h"
00018 #include <DirectXMath.h>
00019 #include <cmath>
00020 #include <cstdlib>
00021
00022 // ============================================================
00023 // カテゴリ 1: データ型のコンポーネント
00024 // ============================================================
00030
00041 struct Health : IComponent {
00042     float current = 100.0f;
00043     float max = 100.0f;
00044
00049     void TakeDamage(float damage) {
00050         current -= damage;
00051         if (current < 0.0f) current = 0.0f;
00052     }
00053
00058     void Heal(float amount) {
00059         current += amount;
00060         if (current > max) current = max;
00061     }
00062
00067     bool IsDead() const {
00068         return current <= 0.0f;
00069     }
00070 };
00071
00081 struct Velocity : IComponent {
00082     DirectX::XMFLOAT3 velocity{ 0.0f, 0.0f, 0.0f };
00083
00090     void AddVelocity(float x, float y, float z) {
00091         velocity.x += x;
00092         velocity.y += y;
00093         velocity.z += z;
00094     }
00095 };
00096
00100 DEFINE_DATA_COMPONENT(Score,
00101     int points = 0;
00102
00107     void AddPoints(int p) {
00108         points += p;
00109     }
00110
00114     void Reset() {
00115         points = 0;
00116     }
00117 );
00118
00122 DEFINE_DATA_COMPONENT(Name,
00123     const char* name = "Unnamed";
00124 );
00125
00130 struct PlayerTag : IComponent {};
00131 struct EnemyTag : IComponent {};
00132 struct BulletTag : IComponent {};
00133
00134 // ============================================================
00135 // カテゴリ 2: シンプルな Behaviour（1 つの機能）
00136 // ============================================================
00142
00153 struct Bouncer : Behaviour {
00154     float speed = 2.0f;
00155     float amplitude = 2.0f;
00156     float time = 0.0f;
00157     float startY = 0.0f;
00158
00164     void OnStart(World& w, Entity self) override {
00165         // 最初に 1 回だけ呼ばれる
00166         auto* t = w.TryGet<Transform>(self);
00167         if (t) {
00168             startY = t->position.y; // 開始位置を記録
00169         }
00170     }
00171
```

```
00178     void OnUpdate(World& w, Entity self, float dt) override {
00179         // 毎フレーム呼ばれる
00180         time += dt * speed;
00181 
00182         auto* t = w.TryGet<Transform>(self);
00183         if (t) {
00184             // sin 波で上下に跳ねる
00185             t->position.y = startY + sinf(time) * amplitude;
00186         }
00187     }
00188 };
00189 
00200 struct MoveForward : Behaviour {
00201     float speed = 2.0f;
00202 
00209     void OnUpdate(World& w, Entity self, float dt) override {
00210         auto* t = w.TryGet<Transform>(self);
00211         if (!t) return;
00212 
00213         // Z 軸方向（前）に進む
00214         t->position.z += speed * dt;
00215 
00216         // 遠くに行ったら削除（オプション）
00217         if (t->position.z > 20.0f) {
00218             w.DestroyEntity(self);
00219         }
00220     }
00221 };
00222 
00232 struct PulseScale : Behaviour {
00233     float speed = 3.0f;
00234     float minScale = 0.5f;
00235     float maxScale = 1.5f;
00236     float time = 0.0f;
00237 
00244     void OnUpdate(World& w, Entity self, float dt) override {
00245         time += dt * speed;
00246 
00247         auto* t = w.TryGet<Transform>(self);
00248         if (!t) return;
00249 
00250         // sin 波でスケールを変化
00251         float scale = minScale + (maxScale - minScale) * (sinf(time) * 0.5f + 0.5f);
00252         t->scale = DirectX::XMFLOAT3{ scale, scale, scale };
00253     }
00254 };
00255 
00265 struct ColorCycle : Behaviour {
00266     float speed = 1.0f;
00267     float time = 0.0f;
00268 
00275     void OnUpdate(World& w, Entity self, float dt) override {
00276         time += dt * speed;
00277 
00278         auto* mr = w.TryGet<MeshRenderer>(self);
00279         if (!mr) return;
00280 
00281         // HSV 風に色を変化（虹色）
00282         float hue = fmodf(time, 1.0f);
00283         mr->color.x = sinf(hue * DirectX::XM_2PI) * 0.5f + 0.5f;
00284         mr->color.y = sinf((hue + 0.333f) * DirectX::XM_2PI) * 0.5f + 0.5f;
00285         mr->color.z = sinf((hue + 0.666f) * DirectX::XM_2PI) * 0.5f + 0.5f;
00286     }
00287 };
00288 
00289 // ============================================================
00290 // カテゴリ 3: 複雑な Behaviour（複数の機能）
00291 // ============================================================
00297 
00308 struct DestroyOnDeath : Behaviour {
00315     void OnUpdate(World& w, Entity self, float dt) override {
00316         // Health コンポーネントを確認
00317         auto* health = w.TryGet<Health>(self);
00318         if (!health) return;
00319 
00320         // 体力が 0 以下なら削除
00321         if (health->IsDead()) {
00322             w.DestroyEntity(self);
00323         }
00324     }
00325 };
00326 
00337 struct RandomWalk : Behaviour {
00338     float speed = 2.0f;
00339     float changeInterval = 2.0f;
00340     float timer = 0.0f;
00341     DirectX::XMFLOAT3 direction{ 1.0f, 0.0f, 0.0f };
```

```
00342
00348     void OnStart(World& w, Entity self) override {
00349         // 最初にランダムな方向を選ぶ
00350         ChooseRandomDirection();
00351     }
00352
00359     void OnUpdate(World& w, Entity self, float dt) override {
00360         timer += dt;
00361
00362         // 一定時間ごとに方向転換
00363         if (timer >= changeInterval) {
00364             timer = 0.0f;
00365             ChooseRandomDirection();
00366         }
00367
00368         // 移動
00369         auto* t = w.TryGet<Transform>(self);
00370         if (!t) return;
00371
00372         t->position.x += direction.x * speed * dt;
00373         t->position.y += direction.y * speed * dt;
00374         t->position.z += direction.z * speed * dt;
00375
00376         // 範囲外に出たら戻す
00377         ClampPosition(t);
00378     }
00379
00380 private:
00384     void ChooseRandomDirection() {
00385         // -1.0 ～ 1.0 のランダムな値
00386         direction.x = (static_cast<float>(rand()) / RAND_MAX) * 2.0f - 1.0f;
00387         direction.y = (static_cast<float>(rand()) / RAND_MAX) * 2.0f - 1.0f;
00388         direction.z = (static_cast<float>(rand()) / RAND_MAX) * 2.0f - 1.0f;
00389
00390         // 正規化（長さを 1 に）
00391         float length = sqrtf(direction.x * direction.x +
00392                              direction.y * direction.y +
00393                              direction.z * direction.z);
00394         if (length > 0.0f) {
00395             direction.x /= length;
00396             direction.y /= length;
00397             direction.z /= length;
00398         }
00399     }
00400
00405     void ClampPosition(Transform* t) {
00406         const float range = 10.0f;
00407         if (t->position.x < -range) t->position.x = -range;
00408         if (t->position.x > range) t->position.x = range;
00409         if (t->position.y < -range) t->position.y = -range;
00410         if (t->position.y > range) t->position.y = range;
00411         if (t->position.z < -range) t->position.z = -range;
00412         if (t->position.z > range) t->position.z = range;
00413     }
00414 };
00415
00426 struct LifeTime : Behaviour {
00427     float remainingTime = 5.0f;
00428
00435     void OnUpdate(World& w, Entity self, float dt) override {
00436         remainingTime -= dt;
00437
00438         // 時間切れで削除
00439         if (remainingTime <= 0.0f) {
00440             w.DestroyEntity(self);
00441         }
00442     }
00443 };
00444
00445 // ============================================================
00446 // カテゴリ 4: マクロを使った簡潔な定義
00447 // ============================================================
00453
00457 DEFINE_BEHAVIOUR(SpinAndColor,
00458     float rotSpeed = 90.0f;
00459     float colorSpeed = 1.0f;
00460     float time = 0.0f;
00461 ,
00462     time += dt * colorSpeed;
00463
00464     // 回転
00465     auto* t = w.TryGet<Transform>(self);
00466     if (t) {
00467         t->rotation.y += rotSpeed * dt;
00468     }
00469
00470     // 色変化
```

```
00471     auto* mr = w.TryGet<MeshRenderer>(self);
00472     if (mr) {
00473         float hue = fmodf(time, 1.0f);
00474         mr->color.x = sinf(hue * 6.28f) * 0.5f + 0.5f;
00475         mr->color.y = cosf(hue * 6.28f) * 0.5f + 0.5f;
00476         mr->color.z = 0.5f;
00477     }
00478 );
00479
00483 DEFINE_BEHAVIOUR(CircularMotion,
00484     float radius = 3.0f;
00485     float speed = 1.0f;
00486     float angle = 0.0f;
00487     float centerY = 0.0f;
00488 ,
00489     angle += speed * dt;
00490
00491     auto* t = w.TryGet<Transform>(self);
00492     if (t) {
00493         t->position.x = cosf(angle) * radius;
00494         t->position.z = sinf(angle) * radius;
00495         t->position.y = centerY;
00496     }
00497 );
00498
00499 // ============================================================
00500 // 使い方の例
00501 // ============================================================
00502 /*
00503
00504 // 例 1: 上下に跳ねる赤いキューブ
00505 Entity cube = world.Create()
00506     .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
00507     .With<MeshRenderer>(DirectX::XMFLOAT3{1, 0, 0})
00508     .With<Bouncer>()
00509     .Build();
00510
00511 // 例 2: 5 秒後に消えるキューブ
00512 Entity temp = world.Create()
00513     .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
00514     .With<MeshRenderer>(DirectX::XMFLOAT3{0, 1, 0})
00515     .Build();
00516
00517 LifeTime lt;
00518 lt.remainingTime = 5.0f;
00519 world.Add<LifeTime>(temp, lt);
00520
00521 // 例 3: 体力システム付きキューブ
00522 Entity enemy = world.Create()
00523     .With<Transform>(DirectX::XMFLOAT3{0, 0, 0})
00524     .With<MeshRenderer>(DirectX::XMFLOAT3{1, 0, 0})
00525     .With<EnemyTag>()
00526     .Build();
00527
00528 Health hp;
00529 hp.current = 50.0f;
00530 hp.max = 50.0f;
00531 world.Add<Health>(enemy, hp);
00532 world.Add<DestroyOnDeath>(enemy, DestroyOnDeath{});
00533
00534 // ダメージを与える
00535 auto* health = world.TryGet<Health>(enemy);
00536 if (health) {
00537     health->TakeDamage(10.0f);
00538 }
00539
00540 */
00541
00542 // ============================================================
00543 // 作成者: 山内陽
00544 // バージョン: v4.0 - 学習用コンポーネント集
00545 // ============================================================
```

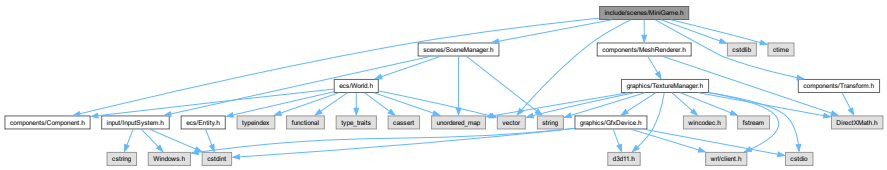
7.33 include/scenes/MiniGame.h ファイル

シンプルなシューティングゲーム

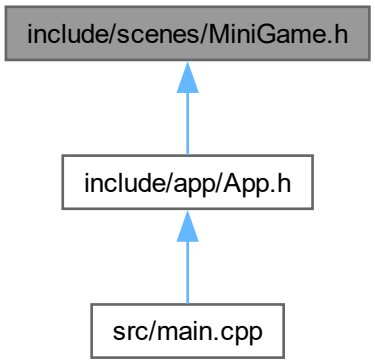
```
#include "scenes/SceneManager.h"
#include "components/Transform.h"
#include "components/MeshRenderer.h"
```

```
#include ”components/Component.h”
#include <cstdlib>
#include <ctime>
#include <vector>
```

MiniGame.h の依存先関係図:



被依存関係図:



クラス

- struct Player
  プレイヤータグ
- struct Enemy
  敵タグ
- struct Bullet
  弾タグ
- struct PlayerMovement
  プレイヤーの移動制御Behaviour
- struct BulletMovement
  弾の移動Behaviour
- struct EnemyMovement
  敵の移動Behaviour
- class GameScene
  シューティングゲームのメインシーン

7.33.1 詳解

シンプルなシューティングゲーム

著者

山内陽

日付

2024

バージョン

4.0

7.33.1.0.1 ゲーム内容:

- プレイヤー (緑キューブ) をA/D キーで左右に移動
- スペースキーで弾を発射
- 敵 (赤キューブ) が自動で降ってくる
- 弾が敵に当たると敵が消滅
- スコアが貯まっていく！

7.34 MiniGame.h

[詳解]

```
00001
00016 #pragma once
00017
00018 #include "scenes/SceneManager.h"
00019 #include "components/Transform.h"
00020 #include "components/MeshRenderer.h"
00021 #include "components/Component.h"
00022 #include <cstdlib>
00023 #include <ctime>
00024 #include <vector>
00025
00026 // ============================================================
00027 // ゲーム用コンポーネント
00028 // ============================================================
00029
00035 struct Player : IComponent {};
00036
00042 struct Enemy : IComponent {};
00043
00049 struct Bullet : IComponent {};
00050
00060 struct PlayerMovement : Behaviour {
00061     float speed = 8.0f;
00062
00069     void OnUpdate(World& w, Entity self, float dt) override {
00070         auto* t = w.TryGet<Transform>(self);
00071         if (!t) return;
00072
00073         // キーボード入力は GameScene で処理
00074         // ここでは位置の制限のみ
00075         if (t->position.x < -8.0f) t->position.x = -8.0f;
00076         if (t->position.x > 8.0f) t->position.x = 8.0f;
00077     }
```

```
00078 };
00079
00089 struct BulletMovement : Behaviour {
00090     float speed = 15.0f;
00091
00098     void OnUpdate(World& w, Entity self, float dt) override {
00099         auto* t = w.TryGet<Transform>(self);
00100         if (!t) return;
00101
00102         t->position.y += speed * dt;
00103
00104         // 画面外に出たら削除
00105         if (t->position.y > 10.0f) {
00106             w.DestroyEntity(self);
00107         }
00108     }
00109 };
00110
00120 struct EnemyMovement : Behaviour {
00121     float speed = 3.0f;
00122
00129     void OnUpdate(World& w, Entity self, float dt) override {
00130         auto* t = w.TryGet<Transform>(self);
00131         if (!t) return;
00132
00133         t->position.y -= speed * dt;
00134
00135         // 画面下に出たら削除
00136         if (t->position.y < -8.0f) {
00137             w.DestroyEntity(self);
00138         }
00139     }
00140 };
00141
00142 // ============================================================
00143 // ゲームシーン
00144 // ============================================================
00145
00161 class GameScene : public IScene {
00162 public:
00170     void OnEnter(World& world) override {
00171         // 乱数シードを設定
00172         srand(static_cast<unsigned int>(time(nullptr)));
00173
00174         // プレイヤーを作成
00175         playerEntity_ = world.Create()
00176             .With<Transform>(
00177                 DirectX::XMFLOAT3{0.0f, -6.0f, 0.0f},   // 画面下部
00178                 DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00179                 DirectX::XMFLOAT3{1.0f, 1.0f, 1.0f}
00180             )
00181             .With<MeshRenderer>(DirectX::XMFLOAT3{0.0f, 1.0f, 0.0f}) // 緑色
00182             .With<Player>()
00183             .With<PlayerMovement>()
00184             .Build();
00185
00186         score_ = 0;
00187         enemySpawnTimer_ = 0.0f;
00188         shootCooldown_ = 0.0f;
00189     }
00190
00197     void OnUpdate(World& world, InputSystem& input, float deltaTime) override {
00198         // プレイヤー移動
00199         UpdatePlayerMovement(world, input, deltaTime);
00200
00201         // 弾の発射
00202         UpdateShooting(world, input, deltaTime);
00203
00204         // 敵の生成
00205         UpdateEnemySpawning(world, deltaTime);
00206
00207         // 衝突判定
00208         CheckCollisions(world);
00209
00210         // ゲームロジックの更新
00211         world.Tick(deltaTime);
00212     }
00213
00221     void OnExit(World& world) override {
00222         // 全エンティティを削除 (イテレータ破壊を回避)
00223         std::vector<Entity> entitiesToDestroy;
00224
00225         world.ForEach<Transform>([&](Entity e, Transform& t) {
00226             entitiesToDestroy.push_back(e);
00227         });
00228
00229         for (const auto& entity : entitiesToDestroy) {
```

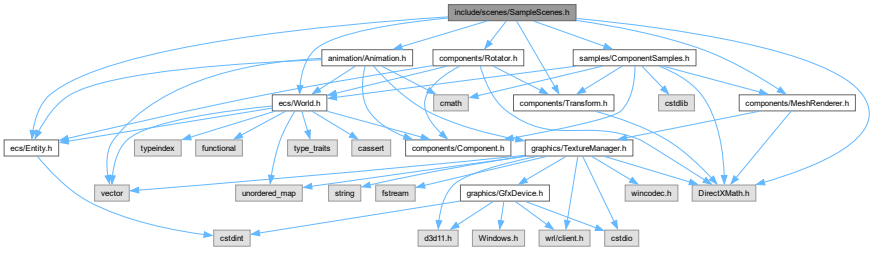
```
00230             world.DestroyEntity(entity);
00231         }
00232     }
00233
00238     int GetScore() const { return score_; }
00239
00240 private:
00247     void UpdatePlayerMovement(World& world, InputSystem& input, float deltaTime) {
00248         auto* playerTransform = world.TryGet<Transform>(playerEntity_);
00249         if (!playerTransform) return;
00250
00251         const float moveSpeed = 8.0f;
00252
00253         // A キーで左移動
00254         if (input.GetKey('A')) {
00255             playerTransform->position.x -= moveSpeed * deltaTime;
00256         }
00257
00258         // D キーで右移動
00259         if (input.GetKey('D')) {
00260             playerTransform->position.x += moveSpeed * deltaTime;
00261         }
00262     }
00263
00270     void UpdateShooting(World& world, InputSystem& input, float deltaTime) {
00271         shootCooldown_ -= deltaTime;
00272
00273         // スペースキーで弾を発射 (クールダウン中は発射できない)
00274         if (input.GetKey(VK_SPACE) && shootCooldown_ <= 0.0f) {
00275             auto* playerTransform = world.TryGet<Transform>(playerEntity_);
00276             if (playerTransform) {
00277                 // プレイヤーの位置から弾を発射
00278                 world.Create()
00279                     .With<Transform>(
00280                         DirectX::XMFLOAT3{playerTransform->position.x, playerTransform->position.y + 1.0f, 0.0f},
00281                         DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00282                         DirectX::XMFLOAT3{0.3f, 0.5f, 0.3f}  // 小さめ
00283                     )
00284                     .With<MeshRenderer>(DirectX::XMFLOAT3{1.0f, 1.0f, 0.0f}) // 黄色
00285                     .With<Bullet>()
00286                     .With<BulletMovement>()
00287                     .Build();
00288
00289                 shootCooldown_ = 0.2f; // 0.2 秒のクールダウン
00290             }
00291         }
00292     }
00293
00299     void UpdateEnemySpawning(World& world, float deltaTime) {
00300         enemySpawnTimer_ += deltaTime;
00301
00302         // 1 秒ごとに敵を生成
00303         if (enemySpawnTimer_ >= 1.0f) {
00304             enemySpawnTimer_ = 0.0f;
00305
00306             // ランダムな位置に敵を配置
00307             float randomX = (rand() % 1600 - 800) / 100.0f; // -8.0 ~ 8.0
00308
00309             world.Create()
00310                 .With<Transform>(
00311                     DirectX::XMFLOAT3{randomX, 8.0f, 0.0f},  // 画面上部
00312                     DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00313                     DirectX::XMFLOAT3{1.0f, 1.0f, 1.0f}
00314                 )
00315                 .With<MeshRenderer>(DirectX::XMFLOAT3{1.0f, 0.0f, 0.0f}) // 赤色
00316                 .With<Enemy>()
00317                 .With<EnemyMovement>()
00318                 .Build();
00319         }
00320     }
00321
00329     void CheckCollisions(World& world) {
00330         // 削除する予定のエンティティリスト (イテレータ破壊を回避)
00331         std::vector<Entity> entitiesToDestroy;
00332
00333         // 弾と敵の衝突をチェック
00334         world.ForEach<Bullet>([&](Entity bulletEntity, Bullet& bullet) {
00335             auto* bulletTransform = world.TryGet<Transform>(bulletEntity);
00336             if (!bulletTransform) return;
00337
00338             // この弾が既に削除予定なら処理をスキップ
00339             for (const auto& e : entitiesToDestroy) {
00340                 if (e.id == bulletEntity.id) return;
00341             }
00342
00343             world.ForEach<Enemy>([&](Entity enemyEntity, Enemy& enemy) {
00344                 auto* enemyTransform = world.TryGet<Transform>(enemyEntity);
```

```
00345                 if (!enemyTransform) return;
00346
00347                 // この敵が既に削除予定なら処理をスキップ
00348                 for (const auto& e : entitiesToDestroy) {
00349                     if (e.id == enemyEntity.id) return;
00350                 }
00351
00352                 // 簡易的な距離判定（円の衝突）
00353                 float dx = bulletTransform->position.x - enemyTransform->position.x;
00354                 float dy = bulletTransform->position.y - enemyTransform->position.y;
00355                 float distance = sqrtf(dx * dx + dy * dy);
00356
00357                 // 衝突したら削除リストに追加してスコア加算
00358                 if (distance < 1.0f) {
00359                     entitiesToDestroy.push_back(bulletEntity);
00360                     entitiesToDestroy.push_back(enemyEntity);
00361                     score_ += 10;
00362                 }
00363             });
00364         });
00365
00366         // イテレーション後にまとめて削除
00367         for (const auto& entity : entitiesToDestroy) {
00368             world.DestroyEntity(entity);
00369         }
00370     }
00371
00372     Entity playerEntity_;
00373     int score_;
00374     float enemySpawnTimer_;
00375     float shootCooldown_;
00376 };
```

7.35 include/scenes/SampleScenes.h ファイル

学習用サンプルシーン集

#include ”ecs/World.h”
#include ”ecs/Entity.h”
#include ”components/Transform.h”
#include ”components/MeshRenderer.h”
#include ”components/Rotator.h”
#include ”animation/Animation.h”
#include ”samples/ComponentSamples.h”
#include <DirectXMath.h>
SampleScenes.h の依存先関係図:



名前空間

- namespace SampleScenes

関数

- Entity SampleScenes::CreateSimpleCube (World &world)
  レベル 1: 最もシンプルなエンティティ
- Entity SampleScenes::CreateRotatingCube (World &world, const DirectX::XMFLOAT3 &position)
  レベル 2: 動きのあるエンティティ
- Entity SampleScenes::CreateBouncingCube (World &world)
  レベル 3: カスタムBehaviour を使う
- Entity SampleScenes::CreateComplexCube (World &world)
  レベル 4: 複数のBehaviour を組み合わせる
- Entity SampleScenes::CreateCubeOldStyle (World &world)
  レベル 5: 従来の方法でエンティティを作成
- void SampleScenes::ModifyEntityExample (World &world, Entity entity)
  レベル 6: コンポーネントの後からの変更
- void SampleScenes::ProcessAllTransforms (World &world)
  レベル 7: 全エンティティに対する処理
- void SampleScenes::ChangeAllColors (World &world)
  全MeshRenderer の色を変更
- void SampleScenes::CreateGridOfCubes (World &world, int rows=3, int cols=3)
  レベル 8: デモシーン作成
- Entity SampleScenes::CreateRainbowCube (World &world)
  練習 1: 虹色に回転するキューブを作成
- Entity SampleScenes::CreateWanderingCube (World &world)
  練習 2: ランダムに動き回るキューブ
- Entity SampleScenes::CreateTemporaryCube (World &world, float lifeTime=5.0f)
  練習 3: 時間経過で消えるキューブ

7.35.1 詳解

学習用サンプルシーン集

著者

山内陽

日付

2024

バージョン

4.0

初学者がコンポーネント指向を段階的に学べるサンプル集です。使い方：各関数をコピーして改造してみよう！

7.36 SampleScenes.h

[詳解]

```
00001
00012 #pragma once
00013
00014 #include "ecs/World.h"
00015 #include "ecs/Entity.h"
00016 #include "components/Transform.h"
00017 #include "components/MeshRenderer.h"
00018 #include "components/Rotator.h"
00019 #include "animation/Animation.h"
00020 #include "samples/ComponentSamples.h"
00021 #include <DirectXMath.h>
00022
00023 namespace SampleScenes {
00024
00025 // ============================================================
00026 // レベル 1: 最もシンプルなエンティティ
00027 // ============================================================
00036
00045 inline Entity CreateSimpleCube(World& world) {
00046     // エンティティを作成（ビルダーパターン）
00047     Entity cube = world.Create()
00048         .With<Transform>(
00049             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},  // 位置
00050             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},  // 回転
00051             DirectX::XMFLOAT3{1.0f, 1.0f, 1.0f}   // スケール
00052         )
00053         .With<MeshRenderer>(DirectX::XMFLOAT3{1.0f, 0.0f, 0.0f}) // 赤色
00054         .Build();
00055
00056     return cube;
00057 }
00058
00059 // ============================================================
00060 // レベル 2: 動きのあるエンティティ
00061 // ============================================================
00069
00079 inline Entity CreateRotatingCube(World& world, const DirectX::XMFLOAT3& position) {
00080     Entity cube = world.Create()
00081         .With<Transform>(
00082             position,
00083             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00084             DirectX::XMFLOAT3{1.0f, 1.0f, 1.0f}
00085         )
00086         .With<MeshRenderer>(DirectX::XMFLOAT3{0.0f, 1.0f, 0.0f}) // 緑色
00087         .With<Rotator>(45.0f) // 毎秒 45 度回転
00088         .Build();
00089
00090     return cube;
00091 }
00092
00093 // ============================================================
00094 // レベル 3: カスタム Behaviour を使う
00095 // ============================================================
00103
00112 inline Entity CreateBouncingCube(World& world) {
00113     Entity cube = world.Create()
00114         .With<Transform>(
00115             DirectX::XMFLOAT3{-3.0f, 0.0f, 0.0f},
00116             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00117             DirectX::XMFLOAT3{0.8f, 0.8f, 0.8f}
00118         )
00119         .With<MeshRenderer>(DirectX::XMFLOAT3{1.0f, 1.0f, 0.0f}) // 黄色
00120         .With<Bouncer>() // 上下に跳ねる（ComponentSamples.h 参照）
00121         .Build();
00122
00123     return cube;
00124 }
00125
00126 // ============================================================
00127 // レベル 4: 複数の Behaviour を組み合わせる
00128 // ============================================================
00136
00145 inline Entity CreateComplexCube(World& world) {
00146     Entity cube = world.Create()
00147         .With<Transform>(
00148             DirectX::XMFLOAT3{3.0f, 0.0f, 0.0f},
00149             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00150             DirectX::XMFLOAT3{1.0f, 1.0f, 1.0f}
00151         )
00152         .With<MeshRenderer>(DirectX::XMFLOAT3{1.0f, 0.0f, 1.0f}) // マゼンタ
00153         .With<Rotator>(30.0f)    // 回転動作
00154         .With<PulseScale>()      // 大きさが変わる（ComponentSamples.h 参照）
```

```
00155         .Build();
00156
00157     return cube;
00158 }
00159
00160 // ============================================================
00161 // レベル 5: 従来の方法でエンティティを作成
00162 // ============================================================
00170
00179 inline Entity CreateCubeOldStyle(World& world) {
00180     // ステップ 1: エンティティを作成
00181     Entity cube = world.CreateEntity();
00182
00183     // ステップ 2: Transform コンポーネントを追加
00184     Transform transform;
00185     transform.position = DirectX::XMFLOAT3{0.0f, -2.0f, 0.0f};
00186     transform.rotation = DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f};
00187     transform.scale = DirectX::XMFLOAT3{0.5f, 0.5f, 0.5f};
00188     world.Add<Transform>(cube, transform);
00189
00190     // ステップ 3: MeshRenderer コンポーネントを追加
00191     MeshRenderer renderer;
00192     renderer.color = DirectX::XMFLOAT3{0.0f, 1.0f, 1.0f}; // シアン
00193     world.Add<MeshRenderer>(cube, renderer);
00194
00195     // ステップ 4: Rotator コンポーネントを追加
00196     Rotator rotator;
00197     rotator.speedDegY = 90.0f;
00198     world.Add<Rotator>(cube, rotator);
00199
00200     return cube;
00201 }
00202
00203 // ============================================================
00204 // レベル 6: コンポーネントの後からの変更
00205 // ============================================================
00213
00222 inline void ModifyEntityExample(World& world, Entity entity) {
00223     // Transform を取得して変更
00224     auto* transform = world.TryGet<Transform>(entity);
00225     if (transform) {
00226         transform->position.y += 1.0f; // Y 座標を 1 上げる
00227         transform->scale = DirectX::XMFLOAT3{2.0f, 2.0f, 2.0f}; // 2 倍の大きさに
00228     }
00229
00230     // MeshRenderer を取得して色を変更
00231     auto* renderer = world.TryGet<MeshRenderer>(entity);
00232     if (renderer) {
00233         renderer->color = DirectX::XMFLOAT3{1.0f, 1.0f, 1.0f}; // 白に変更
00234     }
00235
00236     // Rotator を取得して速度を変更
00237     auto* rotator = world.TryGet<Rotator>(entity);
00238     if (rotator) {
00239         rotator->speedDegY = 180.0f; // 速度を 2 倍に
00240     }
00241 }
00242
00243 // ============================================================
00244 // レベル 7: 全エンティティに対する処理
00245 // ============================================================
00253
00261 inline void ProcessAllTransforms(World& world) {
00262     // 全ての Transform を持つエンティティに対して処理
00263     world.ForEach<Transform>([](Entity entity, Transform& transform) {
00264         // 全てのエンティティを少しずつ上に移動
00265         transform.position.y += 0.01f;
00266     });
00267 }
00268
00276 inline void ChangeAllColors(World& world) {
00277     // 全ての MeshRenderer の色を変更
00278     world.ForEach<MeshRenderer>([](Entity entity, MeshRenderer& renderer) {
00279         // 全てのエンティティを赤っぽくする
00280         renderer.color.x = 1.0f; // R 成分を最大
00281     });
00282 }
00283
00284 // ============================================================
00285 // レベル 8: デモシーン作成
00286 // ============================================================
00294
00304 inline void CreateGridOfCubes(World& world, int rows = 3, int cols = 3) {
00305     const float spacing = 2.5f; // キューブ間の距離
00306
00307     for (int row = 0; row < rows; ++row) {
00308         for (int col = 0; col < cols; ++col) {
```

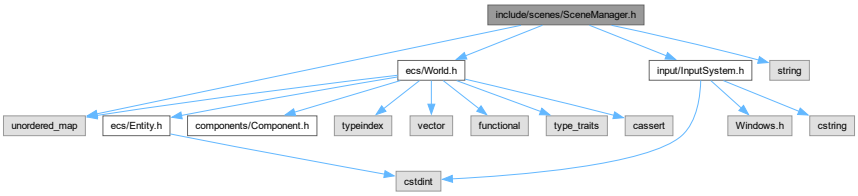
```
00309             // 位置を計算（中心を原点に）
00310             float x = (col - cols / 2.0f) * spacing;
00311             float z = (row - rows / 2.0f) * spacing;
00312 
00313             // 色を計算（位置によって変わる）
00314             float r = static_cast<float>(col) / static_cast<float>(cols - 1);
00315             float b = static_cast<float>(row) / static_cast<float>(rows - 1);
00316 
00317             // キューブを作成
00318             world.Create()
00319                 .With<Transform>(
00320                     DirectX::XMFLOAT3{x, 0.0f, z},
00321                     DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00322                     DirectX::XMFLOAT3{0.8f, 0.8f, 0.8f}
00323                 )
00324                 .With<MeshRenderer>(DirectX::XMFLOAT3{r, 0.5f, b})
00325                 .With<Rotator>(45.0f + static_cast<float>(row * 10 + col * 5))
00326                 .Build();
00327         }
00328     }
00329 }
00330 
00331 // ============================================================
00332 // レベル 9: 練習問題の解答例
00333 // ============================================================
00334 
00343 inline Entity CreateRainbowCube(World& world) {
00344     Entity cube = world.Create()
00345         .With<Transform>(
00346             DirectX::XMFLOAT3{0.0f, 3.0f, 0.0f},
00347             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00348             DirectX::XMFLOAT3{1.0f, 1.0f, 1.0f}
00349         )
00350         .With<MeshRenderer>(DirectX::XMFLOAT3{1.0f, 0.0f, 0.0f})
00351         .With<Rotator>(120.0f) // 速く回転
00352         .With<ColorCycle>()    // 色が変わる（ComponentSamples.h）
00353         .Build();
00354 
00355     return cube;
00356 }
00357 
00366 inline Entity CreateWanderingCube(World& world) {
00367     Entity cube = world.Create()
00368         .With<Transform>(
00369             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00370             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00371             DirectX::XMFLOAT3{0.6f, 0.6f, 0.6f}
00372         )
00373         .With<MeshRenderer>(DirectX::XMFLOAT3{0.8f, 0.3f, 0.9f})
00374         .With<RandomWalk>() // ランダム移動（ComponentSamples.h）
00375         .Build();
00376 
00377     return cube;
00378 }
00379 
00389 inline Entity CreateTemporaryCube(World& world, float lifeTime = 5.0f) {
00390     Entity cube = world.Create()
00391         .With<Transform>(
00392             DirectX::XMFLOAT3{0.0f, 5.0f, 0.0f},
00393             DirectX::XMFLOAT3{0.0f, 0.0f, 0.0f},
00394             DirectX::XMFLOAT3{0.5f, 0.5f, 0.5f}
00395         )
00396         .With<MeshRenderer>(DirectX::XMFLOAT3{1.0f, 0.5f, 0.0f})
00397         .With<Rotator>(200.0f)
00398         .Build();
00399 
00400     // 寿命コンポーネントを追加
00401     LifeTime lt;
00402     lt.remainingTime = lifeTime;
00403     world.Add<LifeTime>(cube, lt);
00404 
00405     return cube;
00406 }
00407 
00408 } // namespace SampleScenes
00409 
00410 // ============================================================
00411 // 使い方の例
00412 // ============================================================
00413 /*
00414 
00415 // App.h の CreateDemoScene() で使う場合:
00416 
00417 void CreateDemoScene() {
00418     // シンプルなキューブ
00419     SampleScenes::CreateSimpleCube(world_);
00420 
```

```
00421     // 回転するキューブ（位置を指定）
00422     SampleScenes::CreateRotatingCube(world_, DirectX::XMFLOAT3{-3, 0, 0});
00423
00424     // 上下に跳ねるキューブ
00425     SampleScenes::CreateBouncingCube(world_);
00426
00427     // 複雑な動きのキューブ
00428     SampleScenes::CreateComplexCube(world_);
00429
00430     // 3x3 のグリッド
00431     SampleScenes::CreateGridOfCubes(world_, 3, 3);
00432
00433     // 練習問題の解答例
00434     SampleScenes::CreateRainbowCube(world_);
00435     SampleScenes::CreateWanderingCube(world_);
00436     SampleScenes::CreateTemporaryCube(world_, 10.0f);
00437 }
00438
00439 */
00440
00441 // ============================================================
00442 // 作成者: 山内陽
00443 // バージョン: v4.0 - 段階的学習用サンプル集
00444 // ============================================================
```

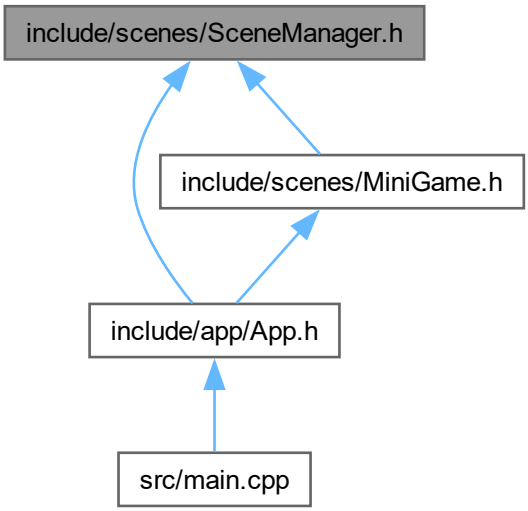
7.37 include/scenes/SceneManager.h ファイル

シーン管理システム

#include ”ecs/World.h”
#include ”input/InputSystem.h”
#include <unordered_map>
#include <string>
SceneManager.h の依存先関係図:



被依存関係図:



クラス

- class IScene
  すべてのシーンの基底クラス
- class SceneManager
  ゲームシーンの切り替えを管理するクラス

7.37.1 詳解

シーン管理システム

著者

山内陽

日付

2024

バージョン

4.0

7.37.1.0.1 役割: ゲームの画面 (シーン) を切り替える

7.37.1.0.2 例: タイトル画面 → ゲーム画面 → リザルト画面

シーン管理の流れ:

1. IScene を継承してシーンを作成
2. SceneManager に登録
3. ChangeScene() で切り替え

7.38 SceneManager.h

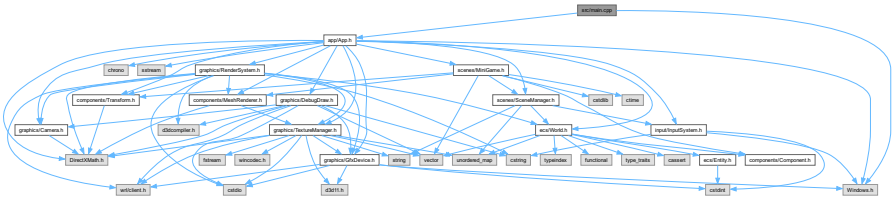
[詳解]

```
00001
00017 #pragma once
00018
00019 #include "ecs/World.h"
00020 #include "input/InputSystem.h"
00021 #include <unordered_map>
00022 #include <string>
00023
00024 // ============================================================
00025 // IScene - シーンの基底クラス
00026 // ============================================================
00027
00073 class IScene {
00074 public:
00078     virtual ~IScene() = default;
00079
00100     virtual void OnEnter(World& world) = 0;
00101
00126     virtual void OnUpdate(World& world, InputSystem& input, float deltaTime) = 0;
00127
00146     virtual void OnExit(World& world) = 0;
00147
00163     virtual bool ShouldChangeScene() const { return false; }
00164
00181     virtual const char* GetNextScene() const { return nullptr; }
00182 };
00183
00184 // ============================================================
00185 // SceneManager - シーン切り替え管理
00186 // ============================================================
00187
00223 class SceneManager {
00224 public:
00235     void Init(IScene* startScene, World& world) {
00236         currentScene_ = startScene;
00237         if (currentScene_) {
00238             currentScene_->OnEnter(world);
00239         }
00240     }
00241
00258     void RegisterScene(const char* name, IScene* scene) {
00259         scenes_[name] = scene;
00260     }
00261
00273     void Update(World& world, InputSystem& input, float deltaTime) {
00274         if (!currentScene_) return;
00275
00276         // 現在のシーンを更新
00277         currentScene_->OnUpdate(world, input, deltaTime);
00278
00279         // シーン遷移チェック
00280         if (currentScene_->ShouldChangeScene()) {
00281             const char* nextSceneName = currentScene_->GetNextScene();
00282             ChangeScene(nextSceneName, world);
00283         }
00284     }
00285
00305     void ChangeScene(const char* sceneName, World& world) {
00306         if (!sceneName) return;
00307
```

```
00308         auto it = scenes_.find(sceneName);
00309         if (it == scenes_.end()) return;
00310
00311         // 現在のシーンを終了
00312         if (currentScene_) {
00313             currentScene_->OnExit(world);
00314         }
00315
00316         // 新しいシーンを開始
00317         currentScene_ = it->second;
00318         if (currentScene_) {
00319             currentScene_->OnEnter(world);
00320         }
00321     }
00322
00327     ~SceneManager() {
00328         // 各シーンを削除
00329         for (auto& pair : scenes_) {
00330             delete pair.second;
00331         }
00332     }
00333
00334 private:
00335     IScene* currentScene_ = nullptr;
00336     std::unordered_map<std::string, IScene*> scenes_;
00337 };
```

7.39 src/main.cpp ファイル

#include <Windows.h>
#include "app/App.h"
main.cpp の依存先関係図:



マクロ定義

- #define WIN32_LEAN_AND_MEAN
- #define NOMINMAX

関数

- int WINAPI WinMain (HINSTANCE hInst, HINSTANCE, LPSTR, int)

7.39.1 マクロ定義詳解

7.39.1.1 NOMINMAX

#define NOMINMAX

7.39.1.2 WIN32_LEAN_AND_MEAN

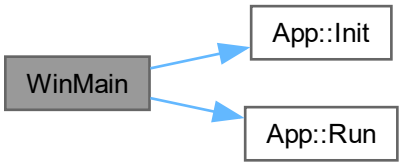
```
#define WIN32_LEAN_AND_MEAN
```

7.39.2 関数詳解

7.39.2.1 WinMain()

```
int WINAPI WinMain (
                HINSTANCE hInst,
                HINSTANCE ,
                LPSTR ,
                int )
```

呼び出し関係図:



Index