# CoGフレームワーク Ver 4.1

2024/07/15

花丸株式会社 花井直人

CoGフレームワークでは、DxLibを使っています。C:\にDxLibをインストールしてください。

■SceneとGameObject

　ゲーム全体は、タイトルや、キャラ選択、ゲームプレイなどの、画面が切り替わります。

プログラムでは、Sceneを切り替えます。

　Scene内に登場するものは、GameObjectと言います。プレイヤーキャラ、敵キャラ、弾など、それぞれが１つのGameObjectになります。

■SceneManager

　シーンの切り替えを管理するのが、SceneManagerです。SceneManagerはnamespaceで作られています。

サンプルでは、TitleScene.cpp::Update()には次のように書いてあります。

|  |
| --- |
| void TitleScene::Update()  {  if (CheckHitKey(KEY\_INPUT\_P))  {  SceneManager::ChangeScene("PLAY");  }  if (CheckHitKey(KEY\_INPUT\_ESCAPE))  {  SceneManager::Exit();  }  } |

　SceneManagerの関数は、SceneManager::を付けて呼び出すことができます。

　‘P’キーを押したときは、ChangeScene(“PLAY”);を呼び出しています。これで、PlaySceneに切り替わります。

　ESCキーを押したときは、Exit()を呼び出しています。これで、アプリ実行を終了します。

　Sceneの切り替えも、アプリ終了も、次のフレームで実行されます。

　次のシーンを、”PLAY“と文字列で指定していますが、これは、クラス名とも、ファイル名とも関係ないので、注意してください。

■SceneFactory

　シーンを切り替える時の文字列と、インスタンスの作成は、SceneFactoryクラスで行っています。

|  |
| --- |
| SceneBase \* SceneFactory::Create(const std::string & name)  {  if (name == "TITLE")  {  return new TitleScene();  }  if (name == "PLAY")  {  return new PlayScene();  }  MessageBox(NULL, ("次のシーンはありません\n" + name).c\_str(),  "SceneFactory", MB\_ICONERROR | MB\_OK);  assert(false);  return nullptr;  } |

　ここで、名前によって、それぞれのSceneを作成する部分が書かれています。Sceneを追加する場合は、ここに書き足してください。

■SceneBase

　Sceneのクラスは、SceneBaseを継承してください。

　PlayScene.hには、次のように書かれています。同じように真似してください。Update()とDraw()が仮想関数になっているので、overrideすれば、毎フレーム呼ばれるようになります。

|  |
| --- |
| #pragma once  #include "../Library/SceneBase.h"  class PlayScene : public SceneBase  {  public:  PlayScene();  ~PlayScene();  void Update() override;  void Draw() override;  private:  }; |

■GameObject

　Sceneの中に登場するものは、すべてGameObjectとして作ります。GameObjectを継承してください。

|  |
| --- |
| #include “../Library/GameObject.h”  class Player : public GameObject  {  Public:  Player();  ~Player();  void Update() override;  void Draw() override;  } |

　new Player()のように、インスタンスを作成できます。コンストラクターで引数を持たせることもできます。

　GameObjectのコンストラクターで、ObjectManagerに登録しているので、Update()とDraw()をoverrideすれば、毎フレーム呼ばれます。

　インスタンスを削除する場合は、deleteでも消すことができますが、自分自身を削除すると、誤動作になる可能性があります。 DestroyMe()を使うとUpdate()の後で削除されるので、安全です。

■ObjectManager

　GameObjectを継承したインスタンスを全て持っています。ここから、毎フレーム、Update()とDraw()を呼び出しています。  
　Sceneが切り替わったときは、GameObjectのインスタンスを削除しています。

　したがって、Update()、Draw()を読んだり、インスタンスを削除したり、をプログラムで書く必要はありません。

■Find関数

　ゲーム中では、他のGameObjectの値を参照したり、関数を読んだりすることがあります。この時は、

|  |
| --- |
| Player\* player = FindGameObject<Player>(); |

とすると、Playerのインスタンスがplayer変数に入ります。

　複数のインスタンスをまとめて取得する場合は、Objectsと複数形にします。

|  |
| --- |
| std::list<Enemy\*> enemies = FindGameObjects<Enemy>(); |

これで、Enemyのすべてのインスタンスがlistで取得できます。

■GameObjectの削除

　不要になったGameObjectは削除します。大抵の場合、自分自身を削除することが多いので、自ら不要になったタイミングで、

|  |
| --- |
| DestroyMe(); |

を呼びます。Update()が終了するときに、削除されます。

　DestroyMe()は、public関数なので、他のクラスから削除することもできます。

|  |
| --- |
| Bomb\* bomb =FindGameObject<Bomb>();  if (bomb != nullptr)  bomb->DestroyMe(); |

とすれば削除されます。

■tagを付ける

　GameObjectにはタグ（文字列）を付けることもできます。GameObjectがタグと一致しているか調べたり、FindGameObjectWithTag()を使って、タグも一致するGameObjectを探すこともできます。

　タグをつける場合は、

|  |
| --- |
| SetTag(“tagString”); |

　タグが一致しているか調べる場合は、

|  |
| --- |
| if (IsTag(“tagString”)) |

　タグが一致するGameObjectを探す場合は、

|  |
| --- |
| Bomb\* bomb = FindGameObjectWithTag<Bomb>(“tagString”); |

　とします。

■描画優先順位

　描画の順番を変えることができます。

|  |
| --- |
| SetDrawOrder(20); |

として、優先順位を設定します。数字が大きいものほど、先に描画されるので、奥に描画されます。描画優先順位の初期値は０です。

■Scene切り替え時のGameObject

　Sceneを切り替える時、すべてのGameObjectが削除されます。通常はこれで問題ありません。