ECS の使い方

目次

1.コンポーネントの作り方

コンポーネントはこのプログラミングにおける最も重要な要素です。データとシステムに分かれます。

※名前空間は省略します

データは以下のように作成します。一言で説明できるような明確なものにしましょう

```
/*!
@brief 座標です,データの型は Vec2 です
*/
struct Position final : public ComponentData
{
	Vec2 val;
	Position() = default;
	Position(const Vec2& v) :val(v) {}
	Position(const float x, const float y) :val(x, y) {}
};
```

コンポーネントシステムは以下のように作成します

```
/*!
@brief コンストラクタで指定したフレーム後に Entity を殺します
*/
class KillEntity final : public ComponentSystem
{
private:
	int cnt_;
public:
	KillEntity(const int span) :cnt_(span) {}
	void update() override
	{
		--cnt_;
		if (cnt_ <= 0)
		{
			entity->destroy();
		}
	}
};
```

エンティティのデータを参照してシステムを作成したい場合は以下のようにします

```
/*!
@brief 座標を Velocity 分動かします
* - Position が必要です
* - Velocity が必要です
*/
class MoveEntity final : public ComponentSystem
{
private:
	Position* pos_;
	Velocity* velocity_;
public:
	void initialize() override
	{
		pos_ = &entity->getComponent<Position>();
		velocity_ = &entity->getComponent<Velocity>();
	}
	void update() override
	{
		pos_->val += velocity_->val;
	}
};
```

2.アーキタイプの作り方

アーキタイプというのはプレイヤーや敵などのオブジェクトのベースになるものです以下のように作成します。

```
Entity* CreateFuga(const char* graphicName, const Vec2 pos,
EntityManager& entityManager_)
{
	auto* fuga = &entityManager_.addEntity();
	fuga->addComponent<Transform>().setPosition(pos.x,pos.y);
	fuga->addComponent<Color>();
	fuga->addComponent<AlphaBlend>();
	fuga->addComponent<SpriteAnimationDraw>(graphicName).setIndex(1);
	fuga->getComponent<SpriteAnimationDraw>().setPivot(Vec2{ 32,32 });
	fuga->addGroup(ENTITY_GROUP::LAYER1);
	return fuga;
}
```

この関数を呼び出した時点でエンティティが作成されます。エンティティ自体はマネージャーによって保存されるので戻り値は保存しなくても構いません

3.エンティティの取得方法

グループごとに返ってきます、戻り値の型は vector になっています

```
auto& layer1 = mana.getEntitiesByGroup(ENTITY_GROUP::LAYER1);
```

4.イベントの作り方

エンティティマネージャーを引数に持つ関数を作成し、イベントマネージャーに登録します。

```
struct Events
{
	static void	Something(ECS::EntityManager& mana)
	{
		//処理
	}
};
//イベント追加
Event::EventManager::Get().addEvent(Scene::SceneManager::State::GAME,
Events::Something);
//イベント更新(更新処理内)
Event::EventManager::Get().update(entityManager_);
```

5.シーンの作り方

以下の抽象クラスを継承し作成します。

```
//!Sceneの基底クラスです
class IScene
{
public:
	virtual ~IScene() = default;
	//!更新処理を行います
	virtual void update() = 0;
	//!描画処理を行います
	virtual void draw() = 0;
	//!リソースの開放を行います
	virtual void release() = 0;
};
```

コンストラクタでエンティティマネージャーを追加してくださいデストラクタはリソース管理の都合上で解放処理として使いません

```
class Game final : public IScene
{
private:
	ECS::Entity* fuga;
	ECS::EntityManager& entityManager_;
public:
	Game(ECS::EntityManager& manager);
	~Game() = default;
	void update() override;
	void draw() override;
	void release() override;
};
```

最終的にはシーンマネージャーが更新を行っています

```
//初期シーンの設定
Scene::SceneManager::Get().changeScene(Scene::SceneManager::State::TITLE,
entityManager_);
Scene::SceneManager::Get().update();//シーン更新
Scene::SceneManager::Get().draw();  //シーン描画
```

6.リソースの管理

リソースマネージャーがハンドルをすべて管理しています

※引数略

```
//同期読み込み
void ResourceManager::GetGraph().loadDiv();
void ResourceManager::GetGraph().load();
void ResourceManager::GetSound().load();
//非同期読み込み
void ResourceManager::GetGraph().loadDivAsync();
void ResourceManager::GetGraph().loadAsync();
void ResourceManager::GetSound().loadAsync();
//非同期読み込み中か調べる
bool ResourceManager::GetGraph().isLoaded();
bool ResourceManager::GetGraph().isLoadedDiv();
bool ResourceManager::GetSound().isLoaded();
//ハンドル取得
int ResourceManager::GetGraph().getHandle();
int ResourceManager::GetSound().getHandle();
int ResourceManager::GetGraph().getDivHandle();
//ハンドル削除
void ResourceManager::GetGraph().removeGraph();
void ResourceManager::GetGraph().removeDivGraph();
void ResourceManager::GetGraph().remove();
//ハンドルチェック
bool ResourceManager::GetGraph().hasHandle();
bool ResourceManager::GetSound().hasHandle();
bool ResourceManager::GetGraph().hasDivHandle();
//非同期読み込みの処理数を取得
static int ResourceManager::GetAsyncLoadNum();
```

アプリケーション終了時にすべて解放します