

Gameplay programming

Lezione 3



De Nadai Mattia - denadaimattia@gmail.com



★ PHYSICS SYSTEM

Questo sistema si occupa di gestire il mondo fisico ovvero la creazione, la rimozione e la simulazione a runtime degli oggetti fisici in game.

★ PHYSICS SYSTEM

Il sistema fornisce le seguenti funzionalità:

- Creazione oggetti fisici
- Rimozione oggetti fisici
- Gestione del PhysicsWorld
- Notifica delle collisioni

★ PHYSICS SYSTEM

```
class PhysicsSystem : public System {
public:
    ID_DECLARATION;
    PhysicsSystem(PhysicsWorldFactory* i_pFactory);
    virtual ~PhysicsSystem();
    virtual void    Init();
    virtual void    Update(real i_fFrametime, real i_fTimestep);

private:
    void NotifyCollisions();
    void CreateObjects(const char* i_szName);
    void DeleteObjects(const char* i_szName);
    PhysicsWorld* m_pPhysicWorld;
    RegistryEventHandler<PhysicsSystem, const char*> m_oRegisterEvent;
    RegistryEventHandler<PhysicsSystem, const char*> m_oUnregisterEvent;
    static const ObjectID PHYSICS_SYSTEM_ID;

};
```

★ PHYSICS SYSTEM

La creazione di un oggetto fisico avviene tramite la definizione di un `PhysicsComponent` che vedremo successivamente.

★ PHYSICS SYSTEM

```
void PhysicsSystem::CreateObjects(const char* i_szName) {  
    const EntityComponentTable* pPhysicsComponentMap= SystemManager::GetSingleton().GetComponentTable(PhysicsComponent::ID);  
    for(EntityComponentTable::const_iterator eclt= pPhysicsComponentMap->begin(); eclt != pPhysicsComponentMap->end(); ++eclt)  
    {  
        const MGDVector<Component*>& vComponents(eclt->second);  
        MGDVector<Component*>::const_iterator itComponent = vComponents.begin();  
        for(; itComponent != vComponents.end(); ++itComponent)    {  
            if((*itComponent)->GetTextName().compare(i_szName) == 0) {  
                PhysicsComponent* pPhysicsComponent= static_cast<PhysicsComponent*>((*itComponent));  
                if(pPhysicsComponent && !pPhysicsComponent->IsInitialize()) {  
                    if(m_pPhysicWorld) {  
                        m_pPhysicWorld->AddObject(pPhysicsComponent->GetPhysicsObject());  
                    }  
                    pPhysicsComponent->SetInit();  
                }  
            }  
        }  
    }  
}
```

★ PHYSICS SYSTEM

La notifica della collisione tra entità fisiche in gioco permette di richiamare funzione definite tramite script

★ PHYSICS SYSTEM

Una collisione può essere gestita da script ridefinendo la funzione “OnCollision”.

I parametri passati sono:

- PhysicsComponent* m_pFirstObject
- PhysicsComponent* m_pSecondObject
- real m_fImpulse;

★ SCRIPT SYSTEM

Questo sistema si occupa di gestire gli script del gioco.

Ogni Script ha delle funzioni che possono essere ridefinite:

- OnEnter: Quando lo script viene creato ed inizializzato
- OnUpdate: Quando lo script è processato a frame-time
- OnExit: Quando lo script viene distrutto

★ SCRIPT SYSTEM

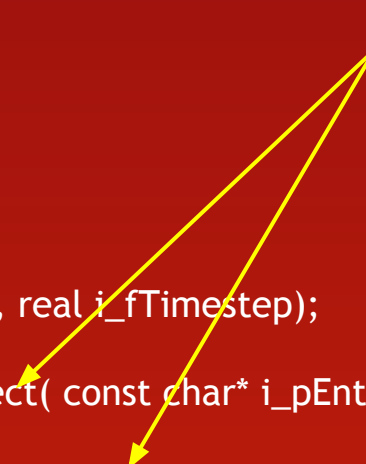
```
class ScriptSystem : public System {
public:
    ID_DECLARATION;
    ScriptSystem();
    virtual ~ScriptSystem();

    virtual void Init();
    virtual void Update(real i_fFrametime, real i_fTimestep);

    static LuaPlus::LuaObject GetLuaObject( const char* i_pEntityName, const char*
i_pComponentId );
    static LuaPlus::LuaObject GetLuaObjectByName( const char* i_pEntityName, const char*
i_pComponentId, const char* i_pComponentName );

private:
    void RegisterCreationObject();
    void RegisterScriptFunction();
};
```

Get Component LuaObject



★ SCRIPT SYSTEM

```
void ScriptSystem::Update( real i_fFrameTime, real i_fTimestep ) {  
    ...  
    ScriptComponent* pScriptComponent= static_cast<ScriptComponent*>((*itComponent));  
    if(pScriptComponent) {  
        if(pScriptComponent->GetState() == ScriptComponent::ScriptState::UNINITILIZE)  
        {  
            pScriptComponent->OnEnter();  
        }  
        else if(pScriptComponent->GetState() == ScriptComponent::ScriptState::RUNNING)  
        {  
            pScriptComponent->OnUpdate(i_fFrameTime, i_fTimestep);  
        }  
        else if(pScriptComponent->IsEnded())  
        {  
            pScriptComponent->OnExit();  
        }  
    }  
    ...  
}
```

★ ANIMATOR SYSTEM

L'Animator system gestisce componenti di tipo `AnimationComponent`.

Permette di gestire una FSM (definita nel componente) che permette di gestire un'animazione tramite script.

Idle -> Run -> Jump -> Idle

★ ANIMATOR SYSTEM

```
class AnimatorSystem : public System
{
public:
    ID_DECLARATION;

    AnimatorSystem();
    virtual ~AnimatorSystem();

    virtual void Init();
    virtual void Update(real i_fFrameTime, real i_fTimestep);
    void UpdateMotions( real i_fTimestep );

    bool ManageSpriteAnimation(real i_fTimestep, MotionSpriteAnimation* i_pMotion, TransformationComponent* i_pTransformationComponent);
    void ManageInterpolationPosition(real i_fTimestep, MotionInterpolationPosition* i_pMotion, TransformationComponent* i_pTransformationComponent);
    void ManageInterpolationRotation(real i_fTimestep, MotionInterpolationRotation* i_pMotion, TransformationComponent* i_pTransformationComponent);
    void ManageInterpolationScale(real i_fTimestep, MotionInterpolationScale* i_pMotion, TransformationComponent* i_pTransformationComponent);
    void ManageAlphaBlend(MotionAlphaBlend* i_pMotion, SpriteComponent* i_pSpriteComponent);

private:
    struct MotionProcess {
        ObjectID m_oType;
        Motion* m_pMotion;
        Component* m_pComponent;
    };

    void DeleteMotions(const char* i_szName);

    MGDMap<ObjectID, MGDVector<MotionProcess>> m_Motions;

    RegistryEventHandler<AnimatorSystem, const char*> m_oUnregisterEvent;
};
```

★ ANIMATOR SYSTEM

La macchina a stati gestita definisce all'interno degli stati degli oggetti chiamati Motion.

Un motion fornisce una determinata funzionalità utilizzabile nell'oggetto da animare

Le tipologie di motion saranno descritte in seguito

★ ANIMATOR SYSTEM

- La creazione dei Motion avviene nell'AnimatorComponent.
- I Motion vengono messi in una coda quando il componente viene creato e processato dal sistema.
- Durante l'update i Motion vengono eseguiti.

```
while(it != m_Motions.end()) {  
    MGDVector<MotionProcess>::iterator itVec = it->second.begin();  
    while(itVec != it->second.end()) {  
        if((*itVec).m_oType == MotionInterpolationPosition::ID) {  
            ManageInterpolationPosition(...);  
            ++itVec;  
        }  
        else if((*itVec).m_oType == MotionInterpolationRotation::ID)  
        {  
            ManageInterpolationRotation(...);  
            ++itVec;  
        }  
        ...  
    }  
}
```

★ GUI SYSTEM

Framework utilizzato: CEGUI



- Facile integrazione con OpenGL e Ogre3D
- Definizione dello stile data-driven
- Definizione di widget data-driven
- Facilità di integrazione con il sistema di input

★ GUI SYSTEM

Questo sistema si occupa della gestione dell'interfaccia utente.

```
class GUISystem : public System {  
public:  
    ID_DECLARATION;  
    GUISystem();  
    virtual ~GUISystem();  
  
    virtual void    Init();  
    virtual void    Update(real i_fFrametime, real i_fTimestep);  
  
    void AddGUIComponent();  
    void DeleteGUIComponent(const char* i_szName);  
  
    ...  
}
```

★ GUI SYSTEM

Gestisce la GUIWindow aggiungendo e rimuovendo i widget definiti all'interno del GUIComponent.

```
void GUISystem::AddGUIComponent() {  
    ...  
    const MGDVector<IGUIWidgets*>& Components(pGUIViewComponent->GetGUIWidgets());  
    for(uint32 uiIndex = 0; uiIndex < Components.size(); ++uiIndex)  
    {  
        CEGUI::Window* pGUIComponent = GetGUIWidget(Components[uiIndex]);  
        if(pGUIComponent) {  
            try {  
                m_pGuiWindow->addChildWindow(pGUIComponent);  
            }  
            catch (CEGUI::Exception& e){}  
        }  
        ...  
    }  
}
```

★ GUI SYSTEM

Ora proviamo ad implementare le seguenti funzioni:

- AddGUIComponent()
 - Gettare il componente GUIViewComponent dal SystemManager
 - Fare un check se è già stato inizializzato
 - Gettare i GUIWidgets dalla GUIViewComponent
 - Tipo di ritorno -> CEGUI::Window*
 - Aggiungere i GuiWidgets del GUIViewComponent alla GuiWindow utilizzando la funzione *addChildWindow*
- DeleteGUIComponent(const char* i_szName)
 - Gettare il componente GUIViewComponent dal SystemManager
 - Fare check se è quello che vogliamo eliminare
 - Gettare i GUIWidgets dalla GUIViewComponent
 - Rimuovere i GuiWidget dalla GuiWindow

★ COMPONENT

Un componente definisce un comportamento che vogliamo dare ad una determinata entità in gioco

Ogni sistema ha il compito di gestire i propri componenti.

★ COMPONENT

I componenti istanziabili sono:

- TransformationComponent
- BaseGfxComponent
- MeshGfxComponent
- GuiComponent
- ScriptComponent
- AnimatorComponent

★ COMPONENT

Ogni component deriva da un'interfaccia chiamata “Component”.

★ COMPONENT

```
class Component {
public:
    ID_DECLARATION;
    Component(std::string i_szOwnerID, bool i_bIsCreatedFromTemplate = false);
    virtual ~Component();

    virtual void Init() = 0;
    virtual bool SetupFromXml(const tinyxml2::XMLElement* pNode);

    void                                SetInit();
    bool                                IsInitialize() const;
    const ObjectID&                     GetOwnerID() const;
    const ObjectID&                     GetName() const;
    const std::string&                  GetTextName() const;

    void                                SetOwnerID(const ObjectID& i_oOwner);
    void                                CreateName();

    ...

    bool IsCreatedFromTemplate() const;
    //LUA
    virtual LuaPlus::LuaObject GetLuaObject();

    void Remove();
    bool IsRemovable() const;

    virtual void CreateFromTemplate(Component* i_pComponent, const ObjectID& i_oOwner) = 0;
    void CreateFromTemplate( const ObjectID& i_oOwner );

protected:
    LuaPlus::LuaObject m_oLuaObject;

private:
    ...
}
```

★ TransformationComponent

- Questo componente si occupa di registrare tutte le informazioni di posizione, rotazione e scalatura.
- Questo componente è inserito di default su ogni entità che creiamo.
- Ha un quaternion per stabilire l'orientamento
- Ha un vector3 per la posizione
- Ha un vector3 per la scalatura
- Viene utilizzato un evento per sincronizzare la fisica con l'oggetto

★ TransformationComponent

Il TransformationComponent deve comunicare direttamente con il sistema fisico in modo da essere aggiornate in base alla simulazione fisica dell'oggetto.

Questo avviene usufruendo del sistema ad eventi implementato.

★ TransformationComponent

Setta la posizione dell'actor

<TransformationComponent>

<Position x="0" y="0" z="0"/>

Setta la rotazione dell'actor

<Scale x="1" y="1" z="1"/>

Setta la scalatura dell'actor

<Rotation degree="90" x="0" y="0" z="1"/>

<LookAt x="0" y="0" z="0"/>

</TransformationComponent>

Utile all'attore camera per creare una matrice tale da guardare un punto specifico.

★ BaseGfxComponent

- Questo componente si occupa di dare una rappresentazione grafica con gli oggetti base
 - CUBE
 - PLANE
 - SPHERE
 - CILINDER
- Deriva da GfxComponent
- Ha una proprietà che ne definisce il materiale

★ BaseGfxComponent

Tipo di primitiva

<BaseGfxComponent>

<PrefabType value="CUBE"/>

Materiale

<MaterialName value="Examples/BeachStones"/>

</BaseGfxComponent>

★ MeshGfxComponent

- Questo componente si occupa di dare una rappresentazione grafica caricando modelli .mesh.
- Deriva da GfxComponent
- Ha una proprietà che ne definisce il materiale

★ MeshGfxComponent

Tipo di mesh

```
<MeshGfxComponent>
```

```
<MeshName value="Base.mesh"/>
```

Materiale

```
<MaterialName value="Phong"/>
```

```
</MeshGfxComponent>
```

★ PhysicsComponent

Questo componente si occupa di settare le proprietà fisiche dell'oggetto.

E' un wrapper del Rigidbody permettendo di definire la collision shape e gestire le collisioni

★ PhysicsComponent

Nella classe
PhysicsComponent nella
funzione SetupFromXML()
trovate tutte le proprietà
configurabili.

```
<PhysicsComponent>  
  <mass value="0"/>  
  <restitution value="1.0"/>  
  <Collision>  
    <shape shape="BOX" x="0.5" y="1" z="2.5"/>  
    <CollisionFlag>  
      <Flag value="KINEMATIC"/>  
    </CollisionFlag>  
  </Collision>  
</PhysicsComponent>
```


★ Component

Abbiamo descritto tutti i componenti principali per poter sviluppare un gioco:

- Posizione degli oggetti in gioco
- Fisica degli oggetti in gioco
- Rendering grafico
- Interazione con l'utente

Come gestiamo le logiche che regolano il punteggio del gioco?

Come gestiamo logiche custom?

★ ScriptComponent

Questo componente ci permette di definire un comportamento custom da attribuire ad un attore

- Il linguaggio di script utilizzato è LUA
- La logica di gioco viene definita custom in uno ScriptComponent
- L'input viene gestito in modo custom utilizzando lo script

★ ScriptComponent

La sintassi di configurazione da XML ha solo una proprietà che definisce il path del file di script

```
<ScriptComponent>
```

```
    <File value="Resources/Pong/Player2Script.lua"/>
```

```
</ScriptComponent>
```

★ ScriptComponent

```
ScriptBall l = GetComponent("Ball","ScriptComponent");
```

Prendo lo script component definito nel file di scena per editarlo (*Nome, Tipo*)

```
function ScriptBall:OnEnter()
```

```
    Logica custom
```

Definisco la logica da eseguire quando l'entità viene istanziata

```
end
```

```
function ScriptBall:OnUpdate(dtMillis)
```

```
    Logica custom
```

Definisco la logica da eseguire quando l'entità viene istanziata

```
end
```

```
function ScriptBall:OnCollision(collisionData)
```

```
    Logica custom
```

Logica eseguita quando avviene una collisione

```
end
```

```
ScriptBall:Setup({ loose = false, timer = 0.0, collisionCount = 0, v = { x = -5, y = 0, z = -5}, points1 = 5, points2 = 5,  
    PhyBall = GetComponent("Ball","PhysicsComponent"),  
    GUI = GetComponent("GameGUI","GUIViewComponent"),  
    TransBall = GetComponent("Ball","TransformationComponent")  
})
```

Funzione che definisce le variabili dello script
(Costruttore)

★ ScriptComponent

Ogni componente ha un membro di tipo LuaObject il quale viene usato negli script.

La funzione GetComponent(...) restituisce l'oggetto Lua che viene creato quando si istanzia un componente.

L'oggetto LUA rispecchia l'oggetto definito in codice nello script

★ ScriptComponent

Lo script viene utilizzato anche per caricare e/o scaricare i layer.

Nella funzione OnEnter dello stato andiamo a chiamare un file script per eseguire l'operazione di load della scena:

```
UnloadActorFile("Resources/2D/Game/EndGameGUI.xml");
```

```
LoadActorFile("Resources/2D/Game/MainMenuGUI.xml");
```

La funzione script è globale e riceve il path del file che contiene la configurazione della scena da caricare.

★ ScriptComponent

Ogni componente espone delle funzioni utilizzabili da script:

- TransformationComponent
 - SetPosition
 - GetPosition
 - ...
- ScriptComponent
 - Setup
- PhysicsComponent
 - ApplyForce
 - SetVelocity
 - GetVelocity
 - ...

✦ ScriptComponent

```
void NomeClass::RegisterScriptFunction()
{
    LuaPlus::LuaObject metaTable = LuaManager::GetSingleton().GetGlobalVars().CreateTable("NomeMetatable");
    metaTable.SetObject("__index", metaTable);

    metaTable.RegisterObjectDirect("NomeFunzioneInLua", (NomeClass*)0, Puntatore a funzione);
}
```


★ ScriptComponent

ESERCITAZIONE

Implementiamo ed esponiamo le seguenti funzionalità del TransformationComponent:

SetPosition

GetPosition

Translate

LookAt

Rotate

GetRotateX

GetRotateY

GetRotateZ

Per convertire un LuaObject in Vector3 e viceversa utilizzare le seguenti funzioni:

LuaUtilities::ConvertLuaObjectToVec3

LuaUtilities::ConvertVec3ToLuaObject

★ GUIViewComponent

Questo componente definisce la User Interface della scena.

- Ha una collezione di Widget

```
const MGDVector<IGUIWidgets*>& GetGUIWidgets() const;
```

✦ Widget

Un widget, è un componente grafico di una interfaccia utente di un programma, che ha lo scopo di facilitare all'utente l'interazione con il programma stesso.

Wikipedia

✦ Widget

Ogni widget deriva da una classe base chiamata IGUIWidgets.

E' un wrapper dei widget di CEGUI.

★ Widget

I widget che sono stati implementati nell'engine attualmente sono:

- StaticText
- Slider
- Button

✦ Widget

La creazione avviene tramite una funzione del GUISystem:

```
CEGUI::Window* GUISystem::GetGUIWidget(IGUIWidgets* i_oGUIWidget) {
    if(i_oGUIWidget)
    {
        CEGUI::Window* pObject = m_pGUIWindowManager->createWindow(i_oGUIWidget->GetStyle(), i_oGUIWidget->GetName());
        if(pObject)
        {
            std::string& szType(i_oGUIWidget->GetType());
            if(strcmp(szType.c_str(),"Button") == 0)
            {
                return Button::Create(i_oGUIWidget, pObject);
            }
            else if(strcmp(szType.c_str(),"StaticText") == 0)
            {
                return StaticText::Create(i_oGUIWidget, pObject);
            }
            else if(strcmp(szType.c_str(),"Slider") == 0)
            {
                return Slider::Create(i_oGUIWidget, pObject);
            }
        }
    }

    return NULL;
}
```

★ GUIViewComponent

Script che definisce la logica della UI

Definizione di un widget

Proprietà del widget

```
<GUIViewComponent>
```

```
<Script filename="Resources/Pong/MainMenuGUILogic.lua"/>
```

```
<GUIComponent>
```

```
<Name value="START"/>
```

```
<Text value="START"/>
```

```
<Type value="Button"/>
```

```
<Style value="TaharezLook/Button"/>
```

```
<Size width="0.15" height="0.05"/>
```

```
<Position x="0.43" y="0.45"/>
```

```
</GUIComponent>
```

```
<GUIComponent>
```

```
<Name value="QUIT"/>
```

```
<Text value="QUIT"/>
```

```
<Type value="Button"/>
```

```
<Style value="TaharezLook/Button"/>
```

```
<Size width="0.15" height="0.05"/>
```

```
<Position x="0.43" y="0.55"/>
```

```
</GUIComponent>
```

```
</GUIViewComponent>
```

✦ Widget

Proviamo ad implementare la funzione:

```
LuaPlus::LuaObject GUIViewComponent::GetLuaWidget(const char* i_szName)
```

- Ogni GUIWidget ha un oggetto Lua che si può avere chiamando la funzione `GetLuaObject()` sull'oggetto

✦ Widget

Utilizzando la documentazione di CEGUI ed i widget già creati proviamo ad implementare un nuovo widget scegliendone uno dalla lista presente al seguente link:

http://cegui.org.uk/wiki/Sample_code_for_all_Widgets