

# Learning Unreal Engine

Stefano Zenaro

Settembre 2022



# Indice

<b>1 Setup Iniziale</b>	<b>5</b>
1.1 Installazione Epic Games Launcher . . . . .	5
1.2 Installazione Unreal Engine . . . . .	5
1.3 IDE per C++ . . . . .	5
1.4 Creare un progetto . . . . .	6
<b>2 Introduzione</b>	<b>9</b>
2.1 Interfaccia grafica . . . . .	9
2.1.1 Viewport . . . . .	9
2.1.2 Toolbar . . . . .	12
2.1.3 Outliner panel . . . . .	12
2.1.4 Details panel . . . . .	12
2.1.5 Content Browser/Drawer . . . . .	12
2.1.6 Menu Bar . . . . .	13
2.2 Post-Processing . . . . .	13
2.2.1 Auto-exposure . . . . .	13
2.2.2 PostProcessVolume . . . . .	14
2.3 Creare un nuovo livello . . . . .	15
2.4 Creare un punto di spawn . . . . .	15
2.5 Introduzione ai materiali . . . . .	15
2.5.1 Utilizzare un materiale . . . . .	15
2.5.2 Creare un materiale . . . . .	15
2.5.3 Material Editor . . . . .	16
2.5.4 Parametri e istanze dei materiali . . . . .	18
2.5.5 Master Material . . . . .	18
2.5.6 Impostazioni PBR . . . . .	18
2.6 Texture . . . . .	19
2.6.1 Importare una texture . . . . .	19
2.7 Static Mesh . . . . .	19
2.7.1 Importare una static mesh . . . . .	19
2.8 Asset . . . . .	20
2.8.1 Migrare asset tra progetti . . . . .	20
2.9 Lumen: gestire la luce . . . . .	20
2.9.1 Luce statica . . . . .	20
2.9.2 Tipi di luce . . . . .	21
2.9.3 Impostazioni lumen . . . . .	22
<b>3 Gameplay Framework</b>	<b>25</b>
3.1 Giocatore . . . . .	25
3.2 Game mode . . . . .	25
3.3 Game state . . . . .	25
3.4 Player state . . . . .	26
3.5 Game Instance . . . . .	26
3.6 Modifica comandi . . . . .	26

<b>4 Blueprint</b>	<b>27</b>
4.1 Viewport . . . . .	27
4.2 Event graph . . . . .	27
4.3 Confronto con codice C++ . . . . .	28
<b>5 Programmazione C++ per UE</b>	<b>29</b>
5.1 Classi predefinite . . . . .	30
5.2 UObject . . . . .	31
5.3 Classe Actor . . . . .	32
5.3.1 Modificare il materiale di una mesh a runtime . . . . .	32
5.4 Classe Scene Component . . . . .	32
5.5 Classe Pawn . . . . .	33
5.6 UPROPERTY . . . . .	33
5.7 UFUNCTION . . . . .	35
5.8 Debugging . . . . .	35
5.8.1 Logging . . . . .	35
5.8.2 Messaggi a video . . . . .	36
5.8.3 Figure geometriche . . . . .	37
5.9 SpawnActor . . . . .	38
5.10 Collezioni di oggetti . . . . .	38
5.10.1 TArray . . . . .	38
5.10.2 TMap . . . . .	38
5.11 Enumerazioni . . . . .	39
5.12 Funzioni e macro utili . . . . .	39
5.13 Delegate e eventi custom . . . . .	40
5.14 Timer . . . . .	42
5.15 FString . . . . .	42
<b>6 Plugin</b>	<b>43</b>
6.1 Geometry Script . . . . .	43
6.2 Datasmith . . . . .	43
6.2.1 Datasmith CAD importer . . . . .	43
6.3 MQTT Utilities . . . . .	43
<b>7 Annotazioni</b>	<b>45</b>
7.1 Asset . . . . .	45
7.2 Environment Light Mixer . . . . .	45
7.3 Hotkey utili . . . . .	45
<b>8 Troubleshooting</b>	<b>47</b>
8.1 Visual Studio . . . . .	47
8.2 Packaging . . . . .	47
8.2.1 Errore: the sdk for windows is not installed properly . . . . .	47
8.3 Evitare flickering delle ombre . . . . .	47
8.4 Crash di Unreal Engine . . . . .	48
8.5 Errori di compilazione . . . . .	48
8.5.1 Nested Containers are not supported . . . . .	48
<b>9 Bibliografia</b>	<b>51</b>

# Capitolo 1

## Setup Iniziale

### 1.1 Installazione Epic Games Launcher

Per installare Unreal Engine e' prima necessario scaricare e installare l'epic games launcher.

Una volta installato e' necessario aprirlo ed effettuare il login.

### 1.2 Installazione Unreal Engine

Dall'Epic Games launcher e' possibile installare Unreal Engine dalla sezione relativa (figura 1.1, cliccare su "Unreal Engine" nell'elenco a sinistra).

Nella sezione Unreal Engine sono presenti diverse tab tra cui:

1. Tab "Novita"'; contiene le novita' relative alle nuove versioni di Unreal Engine
2. Tab "Market place": contiene contenuti gratuiti e a pagamento già pronti da poter inserire all'interno dei propri progetti
3. Tab "Biblioteca" (figura 1.1): permette di aprire progetti già creati, aprire Unreal Engine per creare un nuovo progetto ed installare più versioni del motore grafico.

Il tasto "+" di fianco a "VERSIONI MOTORE" permette di installare una nuova versione di Unreal Engine.

Durante l'installazione e' possibile specificare nelle opzioni cosa installare per ridurre le dimensioni dell'installazione.

#### CONSIGLIO

Tra le opzioni e' possibile specificare anche di scaricare l'"Engine Source" e i "simboli dell'editor per il debug" per fare il debugging del codice C++: quest'ultima opzione e' molto pesante (50 GB) ma permette di eseguire il debugging senza andare incontro ad ostacoli (messaggi del tipo "impossibile visualizzare la risorsa, simboli non trovati").

A fine installazione e' possibile cliccare la freccia a destra del pulsante "Avvia" per modificare le impostazioni di Unreal Engine o disinstallarlo.

Sotto alla sezione "VERSIONI MOTORE" e' presente un elenco con tutti i progetti creati.

Più in basso e' possibile vedere gli oggetti acquistati o riscattati gratuitamente dal marketplace.

### 1.3 IDE per C++

Per programmare in C++ e' necessario installare un IDE.

Alcune valide opzioni sono:

- Visual Studio della Microsoft
- Rider for Unreal della JetBrains

Per maggiori informazioni guardare il capitolo Programmazione C++ per UE a pagina 29.

## 1.4 Creare un progetto

Quando si lancia il motore grafico, dopo aver cliccato sul pulsante "Avvia", apparira' il project browser.

Dal project browser (figura 1.2) e' possibile:

- Aprire o creare nuovi progetti
- Dalle categorie e' possibile selezionare un template del progetto che si vuole creare
- Dopo aver selezionato un template, si puo' selezionare se il progetto usera' Blueprint o C++, la qualita' dei preset, se inserire i contenuti iniziali e se attivare il raytracing

### NOTA

I Blueprint permettono di programmare in modo visuale (collegando assieme blocchi mediante l'interfaccia grafica) azioni da eseguire in base agli eventi lanciati ed esposti dal codice C++ di Unreal Engine oppure in base ad altri eventi personalizzati.

E' possibile mescolare Blueprint e codice C++ all'interno di un progetto.

- Impostare il nome del nuovo progetto e crearlo.

### CONSIGLIO

Personalmente preferisco prima creare i progetti da Unreal Engine e successivamente aprire i progetti gia' creati facendo doppio click sul file .uproject contenuto nella cartella del progetto: in questo modo non e' necessario ogni volta aprire l'Epic Games Launcher, avviare il project browser e selezionare il progetto da aprire.

Figura 1.1: Epic Games Launcher: Biblioteca

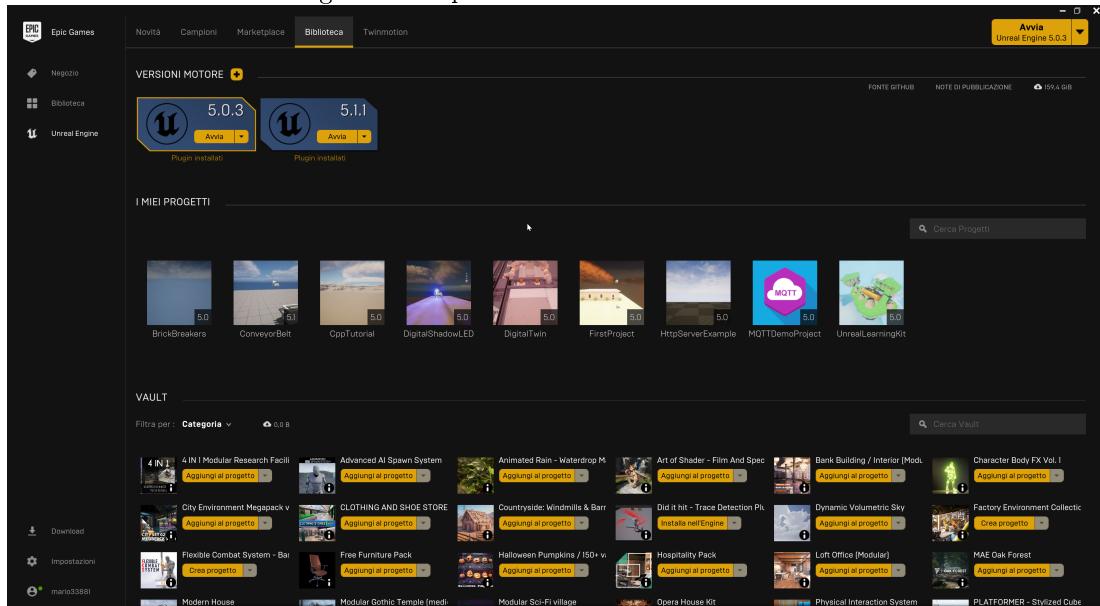
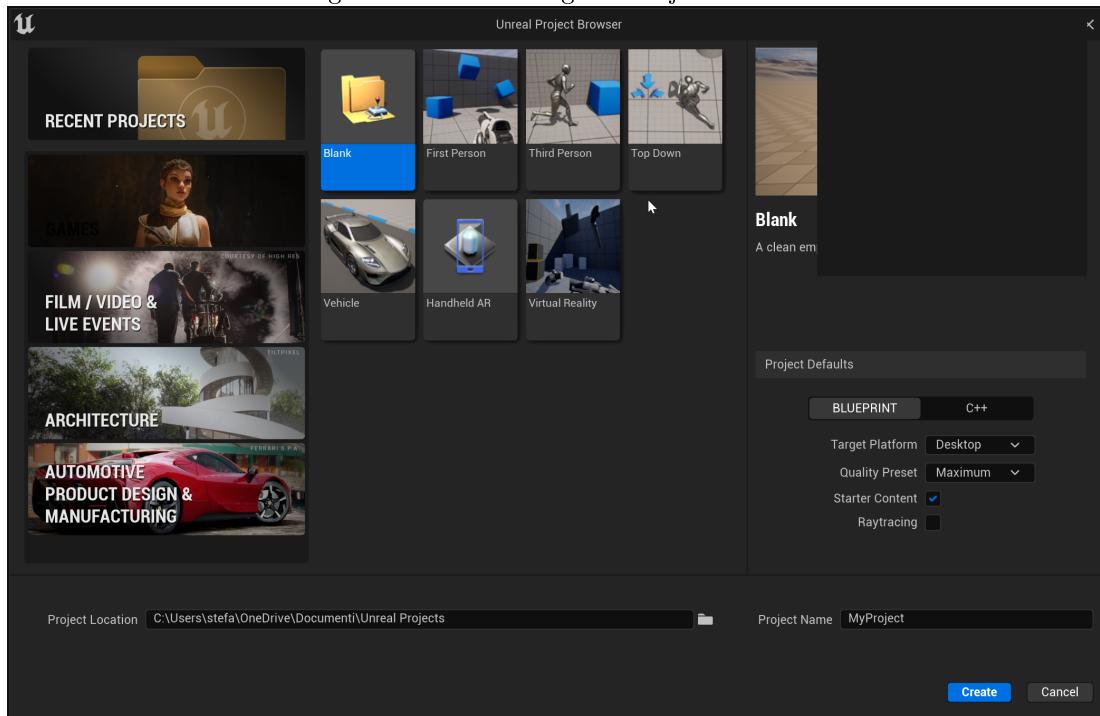


Figura 1.2: Unreal Engine: Project Browser





# Capitolo 2

## Introduzione

### 2.1 Interfaccia grafica

#### NOTA

Tutte le seguenti sotto-finestre possono essere spostate, affiancate, ingrandite o rimpicciolate.

Per ritornare al layout di default delle finestre andare su window, load layout, load default layout.

#### NOTA

Premendo F10 si entra nella "Docked mode" che nasconde tutte le sotto-finestre nella sidebar.

#### 2.1.1 Viewport

La viewport e' la finestra principale (figura 2.2) e permette di inserire, muovere e modificare gli oggetti e di muoversi nel mondo di gioco.

Per muoversi all'interno della viewport occorre tenere premuto il tasto destro e cliccare i tasti W (avanti), A (sinistra), S (indietro), D (destra). Per muoversi verticalmente bisogna premere il tasto E (verso l'alto) o il tasto Q (verso il basso).

Cliccando sul simbolo della telecamera  e' possibile aumentare o diminuire la velocita' del movimento.

#### NOTA

E' anche possibile modificare la velocita' dei movimenti tenendo premuto il tasto destro della viewport e ruotando la rotellina del mouse.

Per teletrasportarsi vicino ad un oggetto si puo' selezionarlo dall'outliner (si trova in alto a destra del viewport) e poi premere il tasto F. (Focus)

Dopo aver selezionato e aver fatto il focus su un oggetto e' possibile:

- Eseguire lo zoom sull'oggetto tenendo premuto il tasto ALT e il tasto destro del mouse e muovendo fisicamente il mouse avanti e indietro oppure ruotando la rotellina.
- Ruotare la visuale attorno all'oggetto tenendo premuto il tasto ALT e il tasto sinistro del mouse e muovendo fisicamente il mouse.

In alto a sinistra, all'interno della viewport, e' presente il tasto "Perspective"  che permette di cambiare prospettiva (ad esempio visualizzare gli oggetti dall'alto).

**NOTA**

Per muoversi dalla prospettiva dall'alto bisogna cliccare il pulsante destro del mouse e trascinarlo. La rotellina del mouse permette di eseguire uno zoom.

Per tornare alla modalita' "Perspective" e' possibile farlo dal tasto di prima oppure premere ALT + G.

Di fianco al tasto "Perspective" e' presente il tasto per modificare la "View mode"  con cui e' possibile vedere il mondo di gioco:

- "Lit": con illuminazione del mondo di gioco
- "Unlit": senza illuminazione (illuminazione "costante" ovunque che permette di vedere i colori "naturali" degli oggetti)
- "Wireframe": permette di vedere spigoli e angoli degli oggetti
- ...

Di fianco al tasto per modificare la "View mode" e' presente il tasto per visualizzare le "Show flags" (figura 2.3): permettono di attivare o disattivare certe tipologie di oggetti.

**NOTA**

Cliccando su "Use defaults" verranno reimpostate le flag originali.

A sinistra del pulsante "Perspective" c'e' un pulsante hamburger da cui si puo' visualizzare la "Game view" (attivabile con il pulsante G). La "Game view" permette di visualizzare direttamente nell'editor il mondo di gioco con gli occhi del giocatore.

**NOTA**

L'hotkey per passare in questa modalita' e' il tasto G.

Con CTRL + L si puo' spostare il sole per vedere i riflessi della luce da varie angolazioni su un oggetto.

**Spostare, ruotare e traslare gli oggetti**

Per spostare gli oggetti bisogna selezionare in alto a destra il pulsante  a destra del pulsante con il cursore. A quel punto e' possibile utilizzare gli assi di traslazione/spostamento (chiamato "gizmo" in Unreal Engine) dell'oggetto selezionato per spostarlo lungo un certo asse.

**NOTA**

E' possibile spostare l'oggetto in modo fluido disattivando la funzione "snap" cliccando sul pulsante griglia  oppure attivarlo per effettuare spostamenti lungo una griglia. E' anche possibile modificare la dimensione della griglia per fare spostamenti piu' o meno precisi (l'unita' di misura e' in centimetri).

Cliccando il cubo al centro dei 3 assi sull'oggetto e' possibile spostare l'oggetto lungo tutte le direzioni.

**NOTA**

Tenendo premuto il tasto shift fa in modo che la telecamera segua l'oggetto che stiamo spostando.

**NOTA**

Premendo sulla tastiera il tasto "Fine" oppure "End" e' possibile far appoggiare un oggetto sulla superficie che si trova al di sotto dell'oggetto stesso.

Per ruotare gli oggetti bisogna selezionare il pulsante di rotazione a destra del pulsante di movimentazione degli oggetti. Utilizzando gli assi e' possibile ruotare l'oggetto.

**NOTA**

Anche la rotazione ha il pulsante per attivare o disattivare la griglia (ed e' possibile modificare la precisione).

Per ridimensionare gli oggetti e' presente il pulsante relativo a destra del pulsante di rotazione . Anche in questo caso e' possibile operare lungo uno dei 3 assi per ridimensionare l'oggetto lungo quella dimensione.

**NOTA**

Anche il ridimensionamento puo' essere fatto lungo una scala di fattori distanziati equivalentemente oppure no cliccando il pulsante relativo .

Cliccando il cubo al centro dei 3 assi e' possibile ridimensionare l'oggetto in modo uniforme lungo tutte le dimensioni.

Tutte e 3 le operazioni possono essere effettuate relativamente all'oggetto stesso oppure relativamente ad un asse globale. (Cliccando sul tasto con il globo a destra del pulsante o cubo a seconda del punto di riferimento attuale)

**Creare un oggetto**

Cliccare il tasto "add" nella toolbar e da li e' possibile aggiungere:

- Una shape
- Una luce
- ...

Un'altra alternativa e' andare nel content browser/drawer (che si apre premendo il tasto relativo in basso a sinistra), selezionare un oggetto e trascinarlo nella viewport.

**Duplicare gli oggetti**

Per duplicare un oggetto e' possibile:

- Cliccare CTRL + D (Duplicate) e poi spostare la copia (l'oggetto copia e l'originale sono sovrapposti).
- Tenere premuto il tasto ALT e contemporaneamente spostare l'oggetto manterra' l'oggetto originale nella sua posizione e spostera' una copia.

**Selezionare piu' oggetti**

Tenendo premuto shift e' possibile cliccare sugli oggetti per selezionarli tutti.

Per de-selezionare un oggetto bisogna tener premuto CTRL e cliccare sull'oggetto da rimuovere.

### Cancellare uno o piu' oggetti

Per cancellare un oggetto e' sufficiente selezionarlo e premere il tasto "Del" oppure "Canc".

Per cancellare piu' oggetti bisogna prima selezionare tutti gli oggetti da cancellare utilizzando shift per selezionare dall'outliner un insieme di oggetti adiacenti oppure tenendo premuto il tasto CTRL e cliccando i singoli oggetti (dalla viewport oppure dall'outliner) che si vuole selezionare.

### 2.1.2 Toolbar

Nella toolbar sono presenti diversi tasti, tra cui:

- Il pulsante "add"  : permette di aggiungere oggetti nel mondo di gioco.
- Il menu' a tendina per selezionare la modalita' tra cui:
  - La "Select Mode": permette di selezionare, spostare e ridimensionare oggetti
  - La "Landscape Mode": permette di creare una landscape
  - La "Foliage Mode": permette di creare fogliame (bisogna avere gli asset disponibili per poter piazzare il fogliame)

La toolbar e' posizionata sopra alla viewport.

### 2.1.3 Outliner panel

E' una lista gerarchica di tutti gli oggetti che appartengono al mondo di gioco. E' posizionata a destra della viewport (in alto).

Si possono selezionare gli oggetti cliccando sull'oggetto nella viewport oppure cliccando sull'oggetto dall'Outliner. E' possibile tenere premuto il tasto CTRL per selezionare piu' oggetti.

Per organizzare gli oggetti si puo' cliccare il tasto destro nell'outliner e selezionare "Create folder" per creare una cartella e poi trascinare gli oggetti al suo interno.

E' anche possibile nascondere temporaneamente gli oggetti cliccando sull'occhio a sinistra dell'oggetto da nascondere. Cliccare di nuovo l'occhio (chiuso) fara' riapparire l'oggetto.

### 2.1.4 Details panel

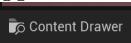
Permette di modificare le proprieta' dell'oggetto selezionato. E' posizionato sotto all'outliner e a destra del viewport (in basso).

Ogni modifica puo' essere annullata premendo CTRL+Z.

#### NOTA

A destra di ogni impostazione modificata e' presente una freccia che permette di resettare l'impostazione al valore di default.

### 2.1.5 Content Browser/Drawer

Contiene tutte le informazioni degli oggetti e del mondo di gioco. Per farlo apparire bisogna cliccare il pulsante in basso a sinistra  chiamato "Content drawer" oppure premere CTRL + barra spaziatrice.

Tutto e' contenuto in sotto-cartelle navigabili facendo doppio click.

E' possibile aprire un asset facendo doppio click sull'asset. Se si fa questa operazione su:

- Una mesh: viene aperto il "static mesh editor" che permette di modificare l'asset.
- Una mappa: viene aperta la mappa. (Verra' chiesto se salvare o eliminare le modifiche effettuate sulla mappa corrente)

Il pulsante "Add" in alto a sinistra all'interno del content drawer permette ad esempio di:

- Cliccare su "Add feature or content pack" per aggiungere nuovi contenuti tra cui il "Starter content" (gli oggetti iniziali)

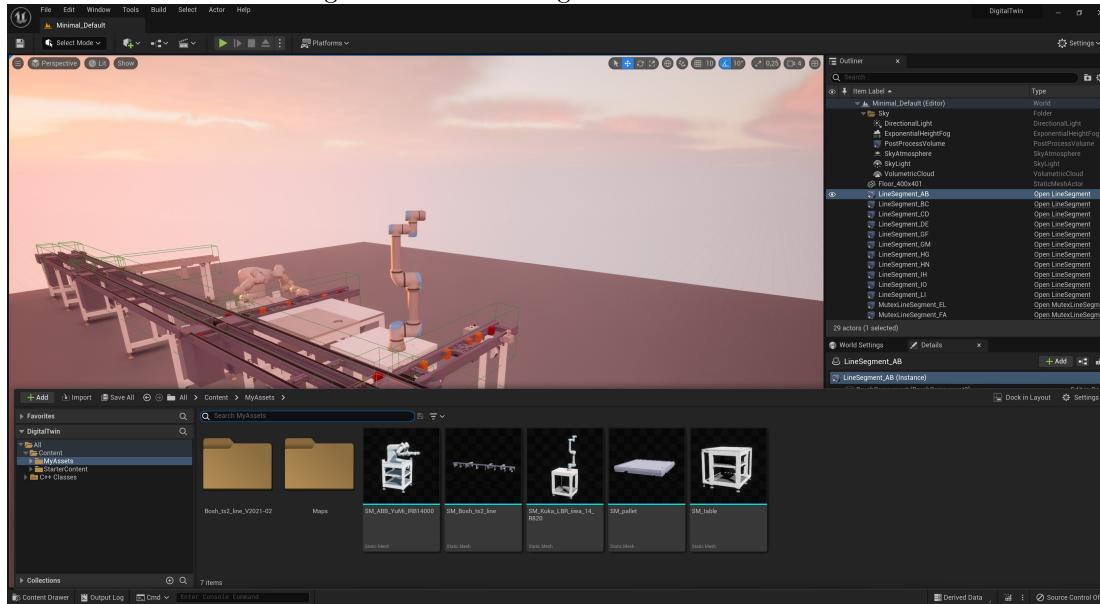
Premendo il tasto "Dock in Layout" in alto a destra nel content drawer permette di tenere sempre aperto il content drawer.

Cliccando il pulsante "Settings", a destra di "Dock in Layout", e' possibile attivare "Show Engine Content" per far apparire nel "Content browser" una cartella contenente tutti gli oggetti di default di Unreal Engine.

Premendo il tasto destro e' possibile:

- Creare una nuova cartella cliccando su "new folder"
- Importare asset cliccando su "import to game"
- Aggiungere modelli e texture da quixel bridge cliccando su "add quixel content"

Figura 2.1: Unreal Engine: Content Browser



### 2.1.6 Menu Bar

La "Menu Bar" e' la barra in alto di fianco al logo di unreal engine.

La "Menu Bar" permette:

- Cliccando su "Window" di vedere una lista di tutte le finestre.

Tra le finestre presenti sotto a "Window" troviamo:

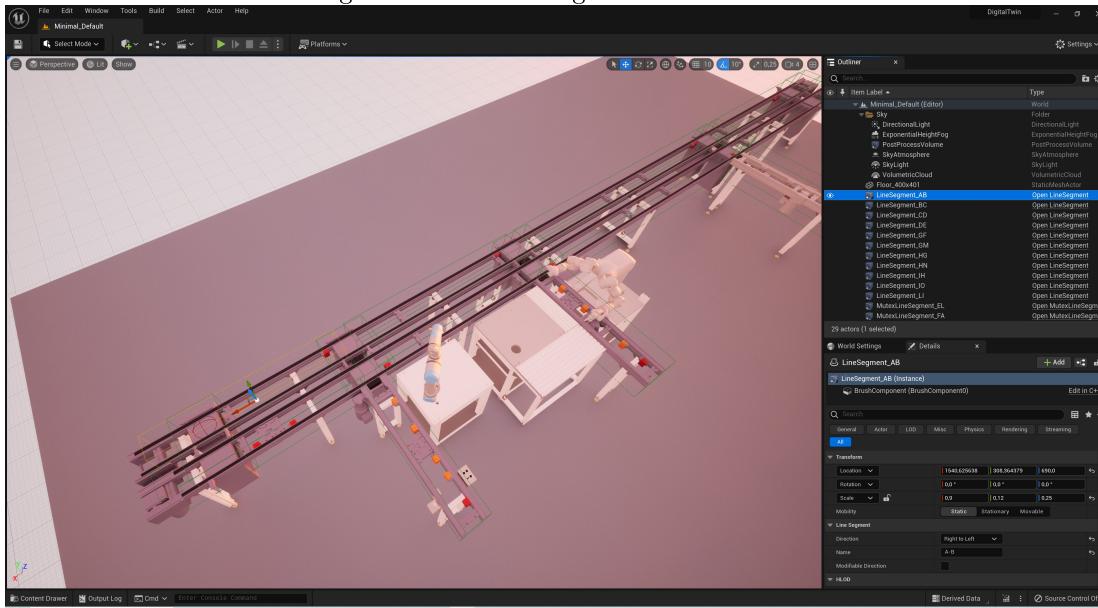
- "World Settings": permette di modificare delle impostazioni della mappa aperta.
  - "Load Layout": permette di impostare le sotto-finestre secondo un certo layout.
- L'impostazione "Default Editor Layout" permette di ritornare al layout originale.

## 2.2 Post-Processing

### 2.2.1 Auto-exposure

Di default Unreal Engine simula l'adattamento degli occhi alla luce e all'ombra modificando la luminosita'.

Figura 2.2: Unreal Engine: Unreal Editor



Per disattivarlo nell'editor e' possibile cliccare sul pulsante "Lit" e de-selezionare "Game settings" sotto a "Exposure".

Sempre all'interno di quel menu' a tendina e' possibile modificare "EV100" per modificare la luminosita'.

Per disattivare l'auto-exposure nel gioco bisogna selezionare l'oggetto "GlobalPostProcessVolume" dall'outliner e poi andare nel "Details panel" e modificare l'impostazione "exposure compensation" dopo aver attivato e impostato l'opzione "Metering Mode" a "Manual".

### 2.2.2 PostProcessVolume

I "PostProcessVolume" permettono di aggiungere degli effetti visivi dopo il rendering. Questi effetti influiscono sulla telecamera e non sul mondo di gioco.

Per creare un PostProcessVolume bisogna andare in "add" nella toolbar, selezionare "volumes" e poi "PostProcessVolume".

Esempio di impostazioni:

- Vignette Intensity: permette di scurire i bordi dello schermo
- (Bloom) Intensity: intensita' dei raggi di luce
- (Color Grading - Global) Saturation

#### NOTA

Impostando la saturazione a zero e' possibile rendere tutto bianco e nero.

- ...

Di default un "PostProcessVolume" ha effetto solo all'interno della sua area. Per rendere l'effetto globale bisogna selezionare l'opzione "Infinite Extent (Unbound)".

#### NOTA

Per vedere l'effetto del "PostProcessVolume" occorre andare sul pulsante "Lit" / "Unlit" e selezionare l'opzione "Game settings"

## 2.3 Creare un nuovo livello

Per creare un nuovo livello e' possibile:

- Andare nel content drawer, fare click con il tasto destro e selezionare "Level" sotto a "Create basic asset".

### NOTA

Il livello sara' vuoto

- Andare su "File", "New level" e selezionare un template predefinito.

## 2.4 Creare un punto di spawn

Per definire un punto di spawn bisogna cliccare su "add", "all classes", "player start".

Se appare l'errore "bad size" significa che il player start collide con un oggetto e che e' necessario spostarlo.

Dopo aver selezionato il player start, premendo il tasto "end"/"fine" della tastiera, e' possibile posizionare il punto di spawn sull'oggetto immediatamente sotto di esso.

Vicino al triangolo verde per iniziare il gioco e' possibile impostare con i 3 puntini verticali il punto di spawn come "default player start".

## 2.5 Introduzione ai materiali

### 2.5.1 Utilizzare un materiale

Per utilizzare un materiale e' sufficiente selezionarlo nel Content Browser e trascinarlo sull'oggetto che deve essere fatto del materiale desiderato.

### NOTA

Per rendere effettive le modifiche effettuate su un materiale utilizzando il Material Editor bisogna compilare il materiale.

### 2.5.2 Creare un materiale

Per creare un materiale occorre:

1. Cliccare il tasto destro in una cartella del content browser
2. Selezionare sotto alla categoria "Create basic asset" la voce "Material"
3. Dare un nome al materiale

### NOTA

Tipicamente un materiale ha il nome che inizia con "M\_ "

Facendo doppio click su un materiale e' possibile aprire il "Material editor" che permette di personalizzare il materiale.

### NOTA

Tenendo premuto il tasto sinistro sulla finestra del "Material editor" e' possibile spostarla e posizionarla di fianco alla tab del livello.

In questo modo, al posto di avere due finestre separate, si avranno due tab come accade nei browser.

### 2.5.3 Material Editor

Il material editor serve per creare o personalizzare un materiale.

Per modificare un materiale e' possibile fare doppio click sul materiale che si trova nel content drawer oppure selezionando un oggetto e' possibile vedere nel details panel i materiali che compongono l'oggetto. Facendo doppio click su uno dei materiali che compone l'oggetto si aprira' il Material Editor.

#### NOTA

E' anche possibile cliccare il pulsante vicino al materiale dal details panel per aprire il content browser nella cartella contenente il materiale e poi fare doppio click sul materiale stesso.

Il Material Editor e' composto da:

- Il "Material Graph": e' formato da nodi interconnessi che permettono di creare il materiale desiderato.

Il nodo di default e' il nodo finale di una catena di nodi.

- La "Viewport" permette di vedere un'anteprima del materiale.

In basso e' possibile decidere se visualizzare il materiale su un modello a forma di cilindro, su una sfera, su un cubo o su una superficie.

- Il "Details Panel" permette di modificare le proprieta' del nodo selezionato.

- La "Palette" (pulsante a destra del material graph) permette di trovare tutti i tipi di nodi inseribili nel Material graph.

#### NOTA

E' anche possibile aprire la "Palette" cliccando il tasto destro nel material graph.

I movimenti possibili all'interno del material graph sono:

- Tenendo premuto il tasto destro e muovendo il mouse e' possibile trascinare il grafo per spostarsi
- La rotellina del mouse permette di eseguire lo zoom
- Cliccare un nodo permette di selezionarlo
- Tenendo premuto il tasto sinistro e muovendo il mouse e' possibile selezionare piu' nodi contemporaneamente
- Tenendo premuto il tasto ALT e cliccando su un punto di giunzione di un filo (pin) con il tasto sinistro permette di cancellare quel filo.
- Tenendo premuto il tasto CTRL e cliccando su un punto di giunzione di un filo (pin) con il tasto sinistro del mouse permette di spostare il filo da un input/output ad un altro.

Tra i nodi presenti nella "Palette" troviamo:

- "Constant3Vector": e' un colore selezionabile dai parametri RGB.

Per applicare il colore costante e' sufficiente collegare il nodo Constant3Vector con il nodo finale nell'uscita chiamata "Base Color".

- "Constant": e' un valore costante.

E' utile ad esempio per impostare se un oggetto e' liscio oppure ruvido collegando il nodo all'output "Roughness". Impostando questo valore vicino a zero si ottiene un materiale riflettente.

**NOTA**

Roughness deve essere un valore tra zero e uno. Zero significa liscio e uno significa ruvido.

Un'altro esempio potrebbe essere impostare l'opzione "Metallic" per rendere l'oggetto metallico o meno.

**NOTA**

Metallic deve essere un valore tra zero e uno. Uno significa metallico.

- "Texture Sample": permette di aggiungere una texture all'oggetto.

Un utilizzo pratico delle texture e' rendere imperfetta la superficie di un oggetto impostando nel details panel l'opzione "Texture" a "T\_Perlin\_Noise\_M" e collegando il valore "RGB" a "Roughness".

**NOTA**

La texture contiene pixel aventi valori tra zero e uno.

- "Linear Interpolate": permette di utilizzare un colore sotto certe condizioni oppure un'altro colore in altre.

Ad esempio e' possibile collegare due colori diversi agli input "A" e "B" e collegare all'input "Alpha" l'output "RGB" di una texture: in questo modo alcune aree saranno colorate del primo colore e le altre nel secondo colore.

**NOTA**

Utile ad esempio per far sembrare sporche alcune superfici.

E' anche possibile utilizzare questo nodo per applicare un colore o una texture. Utile ad esempio per applicare la ruggine con la texture "T\_Metal\_Rust\_D".

**NOTA**

E' possibile aprire la texture facendo doppio click su di essa. Se sembra vuota occorre disattivare l'opacita' cliccando sulla "A".

- "Texture Coordinate": permette di scalare una texture attraverso un nodo "Multiply". La texture coordinate va collegata all'input "A" e un valore costante va collegato all'input "B" di "Multiply".
- "Multiply": permette di moltiplicare due valori tra di loro.
- "Flatten Normal": permette di modificare l'intensita' di una "Normal Map".

**NOTA**

La normal map simula le ombre e le profondita' all'interno dei materiali.

- "StaticSwitchParameter": permette di creare una checkbox che connette un nodo "A" ad un nodo "C" se c'e' la spunta oppure connette il nodo "B" al nodo "C" se non c'e' la spunta.

**NOTA**

True indica che la spunta e' presente, False indica che non e' presenta la spunta.

E' anche possibile importare direttamente le texture trascinandole dal content browser nel material graph.

#### NOTA

E' utile ad esempio per dare una superficie imperfetta ad un materiale (texture "T\_-Perlin\_Noise\_M").

#### NOTA

Per rendere effettive le modifiche effettuate su un materiale utilizzando il Material Editor bisogna compilare il materiale. Per compilare il materiale e' sufficiente cliccare il pulsante "Apply" nella toolbar all'interno del Material editor.

### 2.5.4 Parametri e istanze dei materiali

I parametri permettono di fare modifiche ai materiali senza doverli ricompilare.

Questo permette di risparmiare molto tempo perch' quando un materiale e' estremamente complesso i tempi di compilazione sono lunghi.

Per utilizzare i parametri occorre creare un'istanza dei materiali. Per farlo e' sufficiente aprire il content browser e cercare il materiale, cliccare il tasto destro sul materiale e poi cliccare su "Create Material Instance".

#### NOTA

E' best practice nominare l'istanza come il materiale ma aggiungendo una "I" (Instance) tra la "M" e il "\_": il nome del file inizierà con "MI\_".

L'istanza del materiale avrà accesso soltanto ai parametri impostati all'interno del materiale di cui si è creata l'istanza.

Per aggiungere parametri al materiale bisogna fare doppio click sul materiale (si aprirà il material editor), fare click con il tasto destro su un nodo da trasformare in parametro e cliccare su "Convert to Parameter".

Dal "Details panel" dell'istanza del materiale è possibile modificare i valori dei parametri.

### 2.5.5 Master Material

Un Master Material è un materiale da cui vengono create istanze molto diverse fra di loro.

Generalmente un Master Material ha tutti parametri preconfigurati per NON modificare l'aspetto originale delle texture.

Inoltre anche le texture sono parametri: per convertire una texture in un parametro è sufficiente cliccare con il tasto destro sulla texture e poi cliccare su "Convert to parameter"

### 2.5.6 Impostazioni PBR

PBR, o Physically Based Rendering, è la tecnologia che usa Unreal Engine per simulare un materiale.

Le impostazioni PBR sono:

- Metallic: permette di decidere se è un materiale plastico (valore vicino a zero) o metallico (valore vicino a 1).
- Roughness: permette di decidere se un materiale è lucido e riflettente (valore vicino a zero) o ruvido (Valore vicino a 1).

## 2.6 Texture

### 2.6.1 Importare una texture

Per importare una texture e' sufficiente selezionare i file e trascinarli all'interno del content drawer. I file finiranno nella cartella attualmente aperta nel content drawer.

Esistono diversi tipi di texture:

- Le "Color texture": contengono il colore
- Le "Map"
- Le "normalMap": simulano la profondita' dei materiali (esempio cavita' tra i mattoni) e permettono di avere ombra all'interno della texture.

#### NOTA

Nel Material editor bisogna collegare le normal map al nodo finale attraverso l'output "Normal".

#### NOTA

Per le "color texture" e' importante assicurarsi che sRGB sia attivo. Per verificare se e' attivo fare doppio click sulla texture e guardare nel details panel. Per le altre texture ("map" e "normalMap") e' meglio disattivare sRGB e assicurarsi che sia selezionato il tipo di texture giusto in "Compression settings".

#### NOTA

A volte le texture sono separate (un file per metallic, una per roughness, ...) ma potrebbero anche essere condensate in un unico file seguendo canali di colore diversi (red, green, blue). Quando si vogliono utilizzare texture incluse in un unico file bisogna collegare gli output R, G, B divisi al posto di utilizzare l'output singolo "RGB".

Quando si utilizza una texture in un materiale e' possibile:

- Scalare la texture per uno scalare utilizzando il nodo "Multiply".
- Cambiare la tonalita' della texture utilizzando il nodo "Multiply" per moltiplicare la texture per un colore (Constant3Vector).

## 2.7 Static Mesh

Una static mesh e' un oggetto 3D. Sono composte da vertici, facce e spazio.

Una static mesh ha un materiale che in genere e' composto da piu' texture.

### 2.7.1 Importare una static mesh

Per importare una static mesh in formato FBX occorre trascinare il file .fbx nel content browser.

Lasciando i valori di default verrà creato un nuovo materiale. Se si desidera utilizzare un materiale già esistente bisogna selezionare nel menu' "Material import Method" il valore "Do Not Create Material".

I valori di default faranno importare delle texture. Si puo' specificare di non importare le texture per importarle manualmente.

## 2.8 Asset

### 2.8.1 Migrare asset tra progetti

Per spostare un asset da un progetto Unreal Engine ad un altro e' sufficiente andare nel content browser del progetto contenente una cartella da esportare, cliccare sulla cartella con il tasto destro e premere "Migrate". A quel punto e' possibile de-selezionare gli asset che non si desidera esportare. Dopo aver cliccato su "OK" verra' chiesto in quale cartella importare i file e cliccare su "Select Folder".

Il metodo piu' veloce per trovare il percorso e' trovare la cartella nel content drawer nel progetto in cui si desidera IMPORTARE gli asset, fare click con il tasto destro su una cartella e premere "Show in explorer". Dopo aver fatto questo basta copiare e incollare il percorso nella finestra di selezione della destinazione dell'export descritta precedente e cliccare su "Select folder".

## 2.9 Lumen: gestire la luce

Lumen permette di gestire in tempo reale l'illuminazione della mappa simulando i riflessi della luce. I riflessi contengono parte del colore dell'oggetto riflesso.

L'impostazione "mobility" permette di scegliere se una luce e' "movable", "stationary" oppure "static":

- static: la luce non cambia a runtime, quindi le ombre e la luce restano fisse. Occorre cliccare su "build > build lighting"
- stationary: la luce non si puo' muovere ma le ombre sono dinamiche.
- movable: tutto e' dinamico.

### 2.9.1 Luce statica

Prima che venisse implementato lumen in Unreal Engine le luci erano sempre statiche.

#### NOTA

Per disattivare lumen e' possibile andare nel PostProcessVolume e impostare sotto a "Global Illumination", "Method", l'opzione "None"

E' possibile generare in modo statico le luci e le ombre impostando le luci a "Static" e selezionando la voce "Build", "Build all levels".

Se si spostano gli oggetti dopo aver generato luce statica le luci e le ombre rimarranno fisse.

Per cancellare le "light map" che memorizzano la posizione delle luci e delle ombre statiche bisogna andare in "Window", "World settings", mettere la spunta in "Force No Precomputed Lights" e rifare la build.

E' utile disattivare lumen quando:

- Ci sono risorse molto limitate: lumen richiede piu' risorse della luce statica.
- E' importante la qualita': le luci statiche hanno qualita' piu' alta delle luci lumen.

In generale quando tutti gli oggetti sono immobili e' consigliato utilizzare le luci statiche.

Per passare da lumen alle luci statiche bisogna:

- impostare le luci a static
- in PostProcessVolume occorre disattivare lumen nelle "reflections" e nella "global illumination".

**NOTA**

Per farlo bisogna andare nella sezione relativa, attivare l'opzione "Method" e scegliere l'opzione "None".

- Eseguire la build andando su "build", "build all levels"
- Modificare la qualita' attraverso le impostazioni "lightmass settings" nel PostProcessVolume:
  - Num indirect lighting bounces: numero di salti che fa la luce sulle superfici
  - num skylighting bounces
  - static lighting level scale: riducendolo viene aumentata la qualita'
  - indirect lighting Quality: occorre aumentarlo quando si diminuisce il valore "static lighting level scale".

**NOTA**

Tipicamente il prodotto della "static lighting level scale" per l'"indirect lighting quality" deve fare uno.

Un'altra opzione che migliora la qualita' e' aumentare la "lighting quality" sotto al menu' "build".

- Rieseguire la build. Continuare a modificare le impostazioni per la qualita' e a rieseguire la build finche' non si e' soddisfatti con la qualita' delle luci e delle ombre.

La qualita' della luce viene anche determinata dalla qualita' delle texture.

Premendo il pulsante "Lit" e' possibile andare nel sottomenu' "Optimization viewmodes" e selezionare l'opzione "lightmap density". Il livello verra' colorato di blu, verde e rosso. Blu vuol dire qualita' bassa, verde e rosso qualita' alta.

Selezionare gli oggetti blu, andare nel details panel e attivare l'impostazione "overriden light map res". Aumentando il valore aumentara' la densita' e quindi la qualita' delle texture.

Per diminuire la "pesantezza" delle ombre si puo' selezionare il PostProcessVolume e andare nel a "rendering features", "ambient occlusion" e diminuire l'intensity. Poi aumentare l'"exposure compensation".

Per aggiungere i riflessi bisogna aggiungere una "sphere reflection capture": "add", "visual effects", "sphere reflection capture". La "sphere reflection capture" esegue a 360° uno snapshot del mondo circostante e lo usa per creare i riflessi.

### 2.9.2 Tipi di luce

Tra le luci, che e' possibile posizionare cliccando il pulsante "Add" dalla toolbar, ci sono:

- Le "PointLight": si comportano come lampadine, emettono luce in tutte le direzioni.

Tra le proprieta' piu' importanti troviamo:

- Intensity: intensita' della luce
- Source radius: dimensione dell'emissione di luce. Impostare la dimensione piccola rende le ombre piu' leggere.
- Temperature: permette di definire quanto la luce e' calda o fredda. Un valore basso rende la luce calda, un valore alto rende la luce fredda.

**NOTA**

Se si toglie la spunta su "Use Temperature" verra' ignorata l'impostazione.

- Light color: permette di modificare il colore della luce.

- Indirect lighting intensity: piu' il valore e' alto e piu' la luce rimbalzera' sulle superfici.
- Le "Spotlight": si comportano come le luci di un palcoscenico, emettono luce in una direzione.

Tra le proprietà più importanti troviamo:

- Tutte le proprietà elencate sopra
- "Inner cone angle": permette di definire la "velocità" con cui aumenta la luce allontanandosi dalla fonte
- "Outer cone angle": permette di ingrandire/rimpicciolire l'area illuminata dal fascio di luce
- Le "Rectangle Light": sono simili alle luci spotlight, ma sono rettangolari.

Tra le proprietà più importanti troviamo:

- "Source Width" e "Source height": permettono di aumentare o diminuire la dimensione del rettangolo
- "Barn door angle":
- "Barn door Length": permette di definire la "velocità" con cui aumenta la luce allontanandosi dalla fonte
- Le "Directional Light": simulano la luce del sole. Si diffondono in modo infinito lungo una direzione

Tra le proprietà più importanti troviamo:

- La maggior parte delle proprietà indicate sopra
- "Source angle": permette di rendere le ombre "taglienti" o morbide (valore grande)
- Le "Skylight": "cattura il cielo" e lo proietta.

Per funzionare deve esserci una "Sky atmosphere" e una "Directional Light".

#### NOTA

Se lo skylight non funziona bisogna disattivare e riattivare l'opzione "Affects world" dal details panel.

Tutte le intensità di luce delle altre luci sono relative al valore "intensity scale" dello skylight.

E' possibile cambiare l'impostazione "light color" per cambiare colore alle zone d'ombra.

### 2.9.3 Impostazioni lumen

Per ridurre il rumore delle ombre bisogna aprire il PostProcessVolume e andare nelle impostazioni "Global illumination":

- Final Gather Quality: incrementandolo diminuisce il rumore.

#### NOTA

Costa in termini di prestazioni.

Per aggiungere il cielo bisogna aggiungere una "Sky atmosphere".

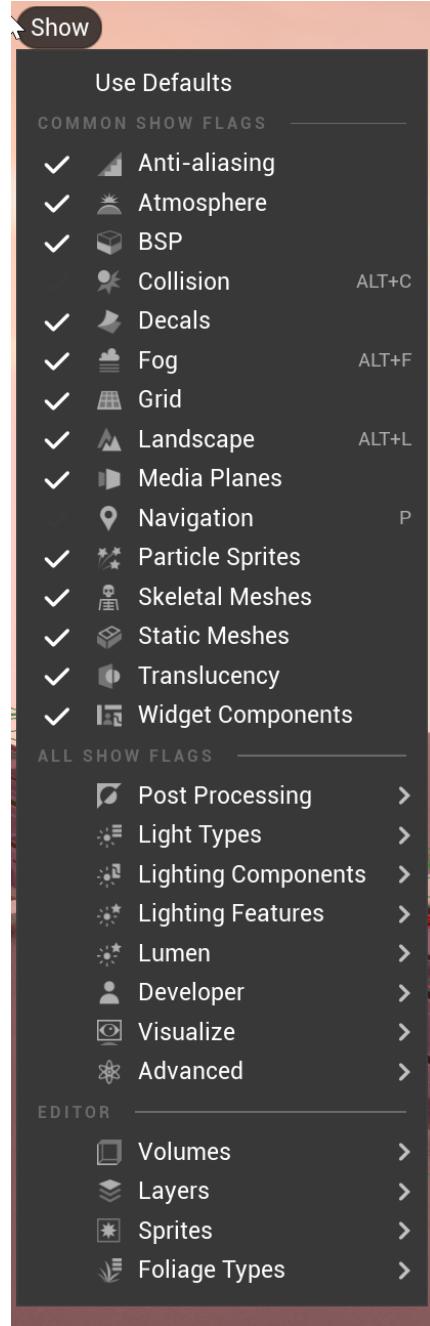
Per nascondere il vuoto nella parte bassa del livello si puo' aggiungere la nebbia aggiungendo con il tasto "add", in "visual effects", l'exponential height fog. Se si vuole creare nebbia ad una distanza maggiore si puo' aumentare il valore della impostazione chiamata "start distance".

Per aggiungere delle nuvole si puo' aggiungere "volumetric cloud" da "visual effects".

Per aumentare i dettagli delle ombre dei piccoli oggetti bisogna attivare nel PostProcessVolume l'opzione "Lumen Scene detail" e aumentarne il valore.

Un'altra opzione nel PostProcessVolume e' cambiare la "Metering Mode" (sotto a "Exposure") a "Manual" per impostare l'esposizione manualmente attraverso l'opzione "Exposure compensation". Si possono anche aumentare i raggi del sole aumentando sotto a "Bloom" il valore della "intensity".

Figura 2.3: Unreal Engine: Unreal Editor Show flags



# Capitolo 3

## Gameplay Framework

### 3.1 Giocatore

In Unreal Engine il giocatore e' composto da due unita':

1. Pawn: personaggio grafico che interagisce con il mondo di gioco
2. Controller: interfaccia tra il giocatore reale e il pawn. Gestisce l'interfaccia grafica e in generale l'input dell'utente.

### 3.2 Game mode

La game mode permette di configurare l'aspetto del pawn e il controller, come inizia un livello, cosa succede quando si smette di giocare... E' la prima cosa che viene istanziata quando viene caricata una mappa e quindi e' ideale per fare setup iniziali.

E' possibile configurare la game mode dell'intero progetto e poi sovrascriverla nei singoli livelli in cui si desidera avere una game mode diversa (se non viene sovrascritta la modalita' di gioco viene ereditata da quella del progetto) dalla finestra:

`Project Settings > Maps & Modes`

#### NOTA

In selected mode e' possibile modificare ulteriori informazioni tra cui "default pawn class", ovvero il modello del giocatore.

Queste informazioni vengono caricate all'inizio di ogni livello e cancellate quando si passa ad un altro livello.

Per sovrascrivere la game mode bisogna modificare l'impostazione "gamemode override" in:

`window > world settings > game mode`

### 3.3 Game state

Permette di memorizzare, processare e sincronizzare dati relativi all'interno gioco (come ad esempio un timer che indica quanto tempo manca al termine della partita)

Queste informazioni vengono caricate all'inizio di ogni livello e cancellate quando si passa ad un altro livello.

### 3.4 Player state

Permette di memorizzare, processare e sincronizzare dati relativi al giocatore (ad esempio la vita rimanente, il punteggio, ...)

Queste informazioni vengono caricate all'inizio di ogni livello e cancellate quando si passa ad un altro livello.

### 3.5 Game Instance

Sono le informazioni che vengono caricate quando viene aperto il gioco e che vengono cancellate quando viene chiuso il gioco.

E' utile memorizzare nella game instance tutti i dati che devono essere mantenuti nel passaggio tra un livello ed un altro.

### 3.6 Modifica comandi

E' possibile configurare i comandi di controllo (mouse, tastiera, joystick, touch, ...) andando in:

```
edit > project settings > engine > input
```

Ci sono due tipi di binding:

- action: quando si preme un pulsante fa qualcosa. Esempio: tasti della tastiera.
- axis: ad ogni frame viene effettuato un qualche check. In genere e' usato per qualcosa di "mobile", come il mouse e i joystick.

# Capitolo 4

## Blueprint

I blueprint di Unreal Engine permettono di programmare in modo visuale, attraverso una interfaccia grafica.

Per ogni blueprint e' possibile vedere e modificare l'aspetto del blueprint dalla finestra "viewport" e modificare gli eventi gestiti dal blueprint in "event graph".

### 4.1 Viewport

In alto a sinistra e' possibile aggiungere componenti al blueprint cliccando sul tasto "Add".

Alcuni componenti utili sono:

- "static mesh": e' l'aspetto che avra' l'oggetto all'interno del gioco
- "arrow" che permette ad esempio di ottenere la sua direzione ed e' solo visibile nel viewport principale e non in gioco
- "box collision" per aggiungere collisioni

### 4.2 Event graph

In event graph e' possibile configurare cosa succede all'istanza del blueprint in base agli eventi che vengono lanciati. E' uno schema a blocchi in cui si specifica quali sequenze di azioni eseguire quando certi eventi sono attivi.

Da questa finestra e' possibile aggiungere variabili sotto a "variables" (e possono essere modificate anche dall'editor nel "details panel") e aggiungere blocchi/nodi per eseguire funzioni quando viene lanciato un evento.

I due eventi di default sono "Event tick" che viene eseguito ad ogni frame e "Event BeginPlay" che viene eseguito quando viene fatto partire il gioco.

Cliccando con il tasto destro e' possibile cercare e piazzare nuovi blocchi/nodi.

Alcuni blocchi utili sono:

- "Get Overlapping Actors" su una box collision: permette di recuperare tutti gli oggetti all'interno della collisione
- "For Each Loop" permette di effettuare un loop for each scorrendo tutti gli oggetti di una lista e per ogni elemento eseguire una serie di operazioni
- "Get Class" permette di recuperare la classe di un oggetto
- "Class is child of" permette di verificare insieme a "Get class" se un oggetto e' figlio di una classe
- "Branch" permette di eseguire il ramo true se una condizione e' vera oppure il ramo false se la condizione e' falsa

- "Add Actor World Offset" permette di spostare un oggetto di una quantita' specificata da un vettore (X, Y, Z) in input relativamente alla sua posizione attuale
- "Multiply" permette di moltiplicare due elementi insieme
- "Break Vector" permette di separare un vettore nelle sue 3 componenti float X, Y, Z
- "Make Vector" permette di costruire un vettore a partire da 3 componenti float X, Y, Z
- "Select Vector" permette di selezionare e restituire un vettore se una condizione e' vera oppure restituirne un altro se la condizione e' falsa.

### 4.3 Confronto con codice C++

Vantaggi dei blueprint rispetto al codice C++:

- Con i blueprint, essendo un "linguaggio" che viene interpretato a runtime, non occorre attendere il tempo di compilazione che e' necessario prima di eseguire il codice C++.  
Questo significa che, in generale, i tempi di sviluppo sono piu' rapidi mediante i blueprint.
- Non e' necessario saper programmare con un linguaggio di programmazione "tradizionale" per aggiungere logica al gioco.

Svantaggi dei blueprint rispetto al codice C++:

- Essendo file binari non e' possibile sfruttare appieno gli strumenti di versioning control come git.  
I sorgenti scritti in codice C++, essendo file di testo, permettono, ad esempio, di vedere ad ogni commit quali porzioni di codice sono state modificate all'interno di un file sorgente.  
Con i blueprint e' solo possibile vedere che quel file e' stato modificato: per rimuovere una certa modifica bisogna sapere quale commit ha effettuato la modifica indesiderata e fare il "rollback" dell'intero blueprint (eliminando anche cio' che si desidererebbe mantenere).
- Le prestazioni dei blueprint, essendo interpretati a runtime, possono risultare peggiori rispetto a codice scritto in C++.

E' consigliato scrivere in codice C++:

- tutto cio' che deve avere alte prestazioni
- file sorgenti la cui scrittura richiede la collaborazione tra piu' programmati, aiutandosi con strumenti come git

E' consigliato utilizzare i blueprint:

- per interfacciare porzioni di codice scritte internamente in C++
- per creare blueprint, possibilmente "piccoli" (logica implementata semplice, pochi nodi, ...), che non richiedono alte prestazioni oppure per creare blueprint che vengono chiamati raramente
- per la configurazione di valori di parametri che possono variare spesso durante la fase di sviluppo
- per permettere a personale che svolge mansioni diverse dalla programmazione (come, ad esempio, i level designer) di configurare facilmente alcuni parametri o di sviluppare blueprint piu' legati alla user experience

#### NOTA

Alcune proprietà che possono essere esposte attraverso i blueprint sono la velocità del giocatore, quanto e' alto un salto, ...

## Capitolo 5

# Programmazione C++ per UE

Per programmare in C++ in Unreal Engine occorre installare Visual Studio e una versione aggiornata di .NET framework SDK.

### CONSIGLIO

Per installare correttamente Visual Studio e' consigliato seguire la guida ufficiale.

### CONSIGLIO

Consiglio anche di provare l'editor Rider for Unreal della JetBrains (gli stessi sviluppatori di IntelliJ IDEA e PyCharm).

E' a pagamento ma e' anche possibile utilizzarlo gratuitamente con la licenza da studente o universitaria.

E' un drop-in replacement di Visual Studio: e' sufficiente aprire il progetto di Unreal Engine e dirgli di utilizzare la soluzione (file .sln) di Visual Studio per cominciare ad utilizzare l'IDE.

Personalmente ho notato che richiede piu' risorse di Visual Studio ma e' piu' integrato nell'ambiente di Unreal Engine: riconosce le keyword delle macro, permette l'eliminazione di file C++ senza dover rigenerare a mano la soluzione mediante il file uproject, segnala variabili che non usano le best practice di Unreal Engine, segnala attributi che potrebbero essere eliminati dal garbage collector di Unreal Engine, file header importati ma non usati, variabili e metodi dichiarati ma mai usati, ...

Quando si crea un nuovo progetto Unreal Engine e' possibile specificare che verra' utilizzato il linguaggio C++ al posto dei blueprint per gestire la logica del gioco. Quando si aprira' il progetto verra' aperto automaticamente Visual Studio.

### NOTA

Se non si apre e' possibile andare in "Tools" e premere il tasto "Open Visual Studio".

### NOTA

E' possibile (anzi, e' consigliato dagli sviluppatori di Unreal Engine) combinare C++ e Blueprint all'interno di un unico progetto.

Se il progetto e' stato configurato per utilizzare i blueprint e' sufficiente creare una classe C++ e Unreal Engine si occupera' di creare tutti i file necessari per programmare in C++.

Per creare una classe C++ e' possibile fare click con il tasto destro nel content browser e poi cliccare su "new C++ class" oppure andare in "Tools" e cliccare su "new C++ Class".

E' consigliato tenere chiuso Unreal Engine mentre si programma in Visual Studio / Rider per evitare problemi con la funzionalita' "Live Coding".

Questa funzionalita' e' pensata per compilare dinamicamente il progetto senza chiudere e riaprire Unreal Engine quando viene effettuata una modifica sul codice C++. Alcuni utenti segnalano che utilizzando questa funzione i progetti Unreal Engine possono corrompersi.

### CONSIGLIO

E' possibile disabilitare il "Live Coding" andando nelle impostazioni del progetto.

In Visual Studio selezionare come metodo di build "Development Editor".

## 5.1 Classi predefinite

Unreal Engine definisce delle classi predefinite a cui e' possibile aggiungere funzionalita' sfruttando l'ereditarieta' delle classi.

Tra le classi predefinite di Unreal Engine troviamo:

- Actor: rappresenta un oggetto che si puo' piazzare nel mondo di gioco. E' possibile legare ad un actor del codice, delle mesh, dell'audio da far partire quando viene lanciato un evento, ...

Dal details panel e' possibile aggiungere componenti all'actor cliccando sul pulsante "Add". I componenti permettono di aggiungere funzionalita' all'actor in modo modulare.

Tra i componenti che si possono aggiungere agli Actor ci sono:

- Arrow: e' una freccia rossa. Di default e' posizionata nell'origine relativa all'actor. Permette di recuperare facilmente la sua posizione dal codice C++.
- Audio: permette di far partire un suono configurabile.

E' anche possibile creare componenti custom.

- Pawn: e' un Actor, puo' essere assegnato ad un giocatore oppure ad una AI. Puo' ricevere input. E' legato ad una camera. Non ha dati collegati ad esso.
- TriggerVolume: e' un oggetto che si puo' piazzare nel mondo di gioco. Riconosce due eventi: quando un oggetto entra all'interno del volume e quando esce. E' possibile legare per ogni evento un metodo con del codice da eseguire quando viene lanciato l'evento.
- UObject: e' l'oggetto di base di Unreal Engine da cui ereditano tutte le altre classi.
- GameInstance: permette di definire una game instance. La game instance memorizza i suoi dati durante tutta l'esecuzione del gioco ed e' accessibile da piu' livelli.

```
// Supponiamo di aver creato (e configurato nell'editor) una nuova
// game instance chiamata DigitalTwinGameInstance: e' possibile richiederla...
UDigitalTwinGameInstance* GameInstance = Cast

```

- Actor Component: e' un componente custom che puo' essere aggiunto ad un actor. Non ha posizione nella gerarchia dei componenti dell'actor: non e' possibile annidare actor component all'interno di altri componenti che appartengono allo stesso actor.

- Scene component: e' un componente custom che puo' essere aggiunto ad un actor. Ha una posizione nella gerarchia dei componenti dell'actor. Al contrario degli Actor component, i Scene component possono essere riposizionati nel mondo ed e' possibile recuperarne le coordinate dal codice.
- Game state: corrisponde al "model" del pattern MVC, contiene i dati, funzioni per memorizzare o caricare i dati, ...
- Game mode: definisce la logica. Definisce eventi in base alle collisioni, chiede al game state di fare qualcosa. Il giocatore interagisce con la game mode.

Premendo il tasto "blueprints" e' possibile accedere alla game mode.

Ci sono 4 impostazioni della game mode che possono essere modificare:

- Game state
- Pawn: sa quali animazioni eseguire, quali oggetti possiede, ...
- HUD: UI all'interno del gioco. Timer, vita, ...
- PlayerController: interfaccia tra l'input e il pawn.

Su "edit", "project settings" e' possibile selezionare la game mode iniziale e la mappa iniziale sia in caso di editing sia in caso di gioco.

- ...

#### ATTENZIONE

NON e' detto che il costruttore (quando presente) venga sempre richiamato alla creazione di ogni istanza di una classe.

Il costruttore viene utilizzato solo per costruire un template delle istanze della classe, non per eseguire operazioni complesse.

Provando ad esempio a richiamare il metodo "NewObject" per istanziare un oggetto di una classe che eredita da UObject si otterra' un puntatore nullptr (si corre quindi facilmente il rischio di far crashare Unreal Engine se si tenta di accedere al puntatore).

## 5.2 UObject

E' l'oggetto di base di Unreal Engine da cui ereditano tutte le altre classi.

Dichiarando che una nostra classe eredita dalla classe UObject permettiamo ad Unreal Engine di gestire vari aspetti dell'oggetto tra cui la gestione della memoria mediante il suo garbage collector.

Gli oggetti che ereditano direttamente da UObject non possono essere piazzati nel mondo ma possono essere istanziati attraverso la chiamata:

```
UClasseCheEreditaDaUObject* NomeVariabile = NewObject<UClasseCheEreditaDaUObject>();
```

La chiamata restituira' un puntatore che permette di gestire l'oggetto. Se viene lasciato pendente il puntatore il garbage collector si occupera' di eliminare l'oggetto.

Metodi utili:

- GetName(): restituisce una FString con il nome dell'oggetto.
- IsA(): permette di verificare se un oggetto e' istanza di una particolare classe.

Esempio d'uso:

```
ActorDiCuiVerificareLaClasse->IsA(ASottoclasseDiActor::StaticClass())
```

ActorDiCuiVerificareLaClasse e' un Actor di cui voglio verificare che appartenga ad una particolare classe. ASottoclasseDiActor e' il nome della classe verificata: se ActorDiCuiVerificareLaClasse e' istanza di ASottoclasseDiActor il metodo restituira' true, altrimenti restituira' false.

### 5.3 Classe Actor

La classe AActor rappresenta un oggetto che puo' essere piazzato nel mondo di gioco.

Metodi utili:

- GetActorLocation(): restituisce un vettore di coordinate X, Y, Z dell'actor nel mondo.
- SetActorLocation(): imposta la posizione di un actor

#### 5.3.1 Modificare il materiale di una mesh a runtime

Modificare a runtime il materiale di una mesh significa permettere di modificare l'aspetto di un oggetto mentre il gioco e' in esecuzione.

Un esempio di applicazione di questa funzionalita' e' creare una luce che lampeggia.

Per prima cosa bisogna creare un componente mesh e aggiungerlo all'actor:

```
// Mesh e' un attributo di tipo class UStaticMeshComponent*
MyMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("My Mesh"));
MyMesh->SetupAttachment(RootComponent);
```

Poi bisogna creare oggetti che rappresentano i materiali:

```
// attributi: (EditAnywhere permette di impostare dall'editor il materiale)
// UPROPERTY(EditAnywhere)
// class UMaterialInterface* OnMaterial;
// UPROPERTY(EditAnywhere)
// class UMaterialInterface* OffMaterial;
OnMaterial = CreateDefaultSubobject<UMaterial>(TEXT("OnMaterial"));
OffMaterial = CreateDefaultSubobject<UMaterial>(TEXT("OffMaterial"));
```

Durante l'esecuzione e' possibile impostare il materiale della mesh richiamando il seguente metodo:

```
MyMesh->SetMaterial(1, OnMaterial);
// oppure
MyMesh->SetMaterial(1, OffMaterial);
```

#### NOTA

Il primo parametro rappresenta l'indice dei materiali della mesh: se la mesh e' composta di un materiale bisognera' sostituire "1" con "0".

Se la mesh e' composta da piu' di un materiale bisogna scegliere l'indice correttamente: visualizzando all'interno dell'editor la mesh e' possibile vedere la lista dei materiali che la compongono.

L'indice inserito come primo parametro corrisponde all'indice del materiale presente in quella lista.

### 5.4 Classe Scene Component

Da "Tools" e' possibile selezionare la voce "New C++ Class".

Da li e' possibile selezionare "Scene Component" e cliccando avanti dargli un nome.

La classe Scene Component ha:

- Un costruttore. Di default viene impostato:

```
PrimaryComponentTick.bCanEverTick = true;
```

Indica che la classe del componente contiene un metodo che deve essere chiamato periodicamente ad ogni tick.

#### NOTA

Un tick e' un frame.

- BeginPlay: (gestisce l'evento BeginPlay) metodo che viene chiamato quando viene eseguito il livello.
- TickComponent: (gestisce l'evento TickComponent) metodo che viene chiamato ogni frame

## 5.5 Classe Pawn

Da "Tools" e' possibile selezionare la voce "New C++ Class".

Da lì e' possibile selezionare "Pawn" e cliccando avanti dargli un nome.

#### NOTA

Visual Studio chiedera' se si desidera ricaricare tutto: dirgli di sì.

Di default una classe che eredita da Pawn ha:

- Un costruttore: viene richiamato quando viene creato l'oggetto della classe.  
Permette di impostare dei valori di default di alcune proprietà.
- Il metodo "BeginPlay": viene chiamato quando viene fatto partire un livello.
- Il metodo "Tick": viene chiamato per ogni frame. Utile per aggiornare le animazioni, per la vita, ...
- Il metodo "SetupPlayerInputComponent": permette di registrare ad ogni input delle azioni.

## 5.6 UPROPERTY

UPROPERTY permette di specificare il comportamento di una proprietà all'interno del motore grafico e dell'editor.

Un'altra funzione MOLTO IMPORTANTE che ha UPROPERTY e' specificare al garbage collector di Unreal Engine come gestire oggetti complessi memorizzati in memoria dinamica.

#### ATTENZIONE

Se si nota che durante l'esecuzione di Unreal Engine avvengono dei crash improvvisi apparentemente immotivati e' possibile che il garbage collector abbia eliminato un oggetto e accedendo al suo puntatore e' avvenuto il crash.

Questo puo' capitare, ad esempio, quando un TArray memorizza puntatori di oggetti: finche' il garbage collector non elimina uno di questi oggetti e finche' non si tenta di accedere alla memoria dell'oggetto eliminato funzionera' tutto. Nel momento in cui avviene l'accesso Unreal Engine crashera'.

Per risolvere il problema e' sufficiente aggiungere "UPROPERTY()" sopra agli attributi (sopra alla dichiarazione del TArray in questo caso).

La sintassi per utilizzare questi modificatori e':

```
UPROPERTY(<modificatore 1>, ...)
<proprietà '/attributo>
```

Ecco un elenco di alcuni modificatori:

- **VisibleAnywhere:** permette di rendere una proprieta' visible in tutte le finestre (ma NON la rende modificabile).

Dopo aver aggiunto questo modificatore ed aver selezionato dall'outliner un oggetto e' possibile vedere nel details panel il valore della proprieta' esposta.

- **EditAnywhere:** permette di visualizzare e di modificare una proprieta' (utile per modificare il valore di interi, booleani, float, ... ).

Dopo aver aggiunto questo modificatore ed aver selezionato dall'outliner un oggetto e' possibile vedere E modificare nel details panel il valore della proprieta' esposta.

- **VisibleDefaultsOnly:** come VisibleAnywhere ma permette di visualizzare la proprieta' solo all'interno dell'editor dei blueprint.

- **EditDefaultsOnly:** come EditAnywhere ma permette di visualizzare e modificare la proprieta' solo all'interno dell'editor dei blueprint.

- **VisibleInstanceOnly:** come VisibleAnywhere ma permette di visualizzare la proprieta' solo quando e' stata creata l'istanza di un oggetto nel mondo.

- **EditInstanceOnly:** come EditAnywhere ma permette di visualizzare e modificare la proprieta' solo quando e' stata creata l'istanza di un oggetto nel mondo.

- **Category:** permette di specificare sotto a quale categoria nel details panel sara' presente il campo che visualizza il valore della proprieta'.

Esempio di sintassi:

```
UPROPERTY(EditAnywhere, Category = "MyCategory")
float speed = 10.0f;
```

#### NOTA

E' possibile mettere piu' proprieta' legate logicamente secondo un qualche criterio all'interno della stessa categoria per organizzare le proprieta'.

- **meta:** permette di aggiungere metadati tra cui:

- **AllowPrivateAccess:** permette di visualizzare una proprieta' nell'editor anche se la proprieta' e' dichiarata private nel codice.

Esempio di sintassi:

```
UPROPERTY(EditAnywhere, meta = (AllowPrivateAccess = "true"))
float speed = 10.0f;
```

- **EditCondition:** permette di modificare il valore di una proprieta' solo se una condizione e' vera.

Esempio d'uso:

```
UPROPERTY(EditAnywhere, Category = "Test", meta = (EditCondition = "Variable > 0.0"))
float speed = 10.0f;
```

Variable deve essere piu' grande di zero per poter modificare il valore di speed.

– ...

- **BlueprintReadOnly:** rende la proprieta' accessibile dai blueprint

- **BlueprintReadWrite:** rende la proprieta' accessibile e modificabile dai blueprint

- ...

**ATTENZIONE**

I modificatori di visibilita' e di edit non possono essere usati contemporaneamente. Inoltre e' solamente possibile specificare un unico modificatore di tipo edit/visibilita' per UPROPERTY.

## 5.7 UFUNCTION

UFUNCTION permette di rendere una funzione visibile dai blueprint.

Un'altra funzionalita' utile di UFUNCTION e' di permettere ad un metodo di essere chiamato quando viene lanciato un evento (callback).

Tra i parametri di UFUNCTION ci sono:

- BlueprintPure: restituisce immediatamente un valore dati dei parametri
- BlueprintCallable: restituisce un valore quando viene richiamata
- BlueprintImplementableEvent: permette di implementare una funzione attraverso un blueprint permettendo di organizzare un blueprint in sotto-blueprint.

**NOTA**

Non e' possibile implementare la funzione in codice C++

- BlueprintNativeEvent: permette di implementare la funzione sia in codice C++, sia sovrascriverla con i blueprint.

Per aggiungere l'implementazione nel codice C++ occorre aggiungere al nome della funzione "\_Implementation".

**NOTA**

Per richiamare la funzione da un'altra funzione nel codice NON bisogna aggiungere "\_Implementation".

## 5.8 Debugging

### 5.8.1 Logging

Per scrivere messaggi di log e' possibile usare la sintassi:

```
UE_LOG(
    <categoria>,
    <verbosita'>,
    <messaggio>,
    [<parametri>, ...]
);
```

Dove:

- categoria: e' il tipo di log. Tipicamente si usa "LogTemp" per indicare che il log viene memorizzato su un file temporaneo.
- verbosita': specifica la gravita' di un messaggio. Tra le verbosita' troviamo:
  - Warning
  - Error
  - Fatal

- messaggio: e' il messaggio da visualizzare. La sintassi da usare e':

```
TEXT("messaggio da visualizzare nei log")
```

All'interno della stringa e' possibile inserire dei valori indicando il simbolo di percentuale seguito da un carattere che indica il tipo della variabile di cui visualizzare il valore:

```
TEXT("Su questa riga apparira' un numero intero: %d")
TEXT("Su questa riga apparira' un float: %f")
TEXT("E qui una stringa: %s")
```

#### NOTA

I caratteri da inserire dopo il simbolo del percentuale sono gli stessi della programmazione in C per la funzione printf.

Le variabili vanno indicate come quarto, quinto, ... n-esimo parametro di UE\_LOG.

- parametri: variabili da cui recuperare i valori per inserirli nel testo.

#### ATTENZIONE

Per le stringhe bisogna prefissare al nome della variabile l'asterisco "\*".

Esempio di messaggi di log:

```
UE_LOG(LogTemp, Warning, TEXT("Su questa riga apparira' un numero intero: %d"), IntVariable);
UE_LOG(LogTemp, Error, TEXT("Su questa riga apparira' un float: %f"), FloatVariable);
UE_LOG(LogTemp, Fatal, TEXT("E qui una stringa: %s"), *StringVariable);
```

### 5.8.2 Messaggi a video

Sintassi:

```
GEngine->AddOnScreenDebugMessage(<id>, <tempo_secondi>, <colore>, <messaggio>);
```

Dove:

- id: identificativo univoco che permette di evitare di far apparire lo stesso messaggio piu' volte
- tempo\_secondi: tempo in secondi da aspettare prima di visualizzare il messaggio (float)
- colore: colore del testo.

Esempio:

```
FColor::Black // colore nero
```

- messaggio: messaggio da visualizzare

Esempio:

```
FString::Printf(TEXT("Questa e' una stringa: %s"), *StringVariable)
```

Esempio:

```
GEngine->AddOnScreenDebugMessage(
    1,
    100.0f,
    FColor::Black,
    FString::Printf(TEXT("Questa e' una stringa: %s"), *StringVariable)
);
```

### 5.8.3 Figure geometriche

E' possibile disegnare delle figure geometriche sull'oggetto che si sta debuggando.

Bisogna includere la libreria:

```
#include "DrawDebugHelpers.h"
```

Tra le figure geometriche troviamo:

- DrawDebugSphere: disegna una Sfera.

#### ESEMPIO

```
DrawDebugSphere(
    // mondo
    GetWorld(),
    // posizione
    GetActorLocation() + FVector(0,0,200.0f),
    // raggio
    20.0f,
    // numero segmenti della sfera
    20,
    // colore
    FColor::Orange,
    // righe persistenti
    true,
    // tempo di durata
    100.0f,
    // priorita'/profondita'
    -1.0f,
    // spessore
    2.0f
);
```

- DrawDebugBox: disegna un parallelepipedo.

#### ESEMPIO

```
DrawDebugBox(
    // mondo in cui creare il parallelepipedo
    GetWorld(),
    // posizione del parallelepipedo
    GetActorLocation(),
    // dimensioni del parallelepipedo
    GetComponentsBoundingBox().GetExtent(),
    // colore
    FColor::Red,
    // righe persistenti
    true,
    // tempo di durata
    999,
    // priorita'/profondita'
    0,
    // spessore
    2
);
```

## 5.9 SpawnActor

Per creare l'istanza di un Actor (o di una classe derivata da AActor) a tempo di runtime e' possibile chiamare la funzione `SpawnActor`.

```
GetWorld()->SpawnActor(<parametri>);
```

`SpawnActor` restituisce un puntatore che permette di richiamare i metodi dell'actor.

## 5.10 Collezioni di oggetti

Unreal Engine mette a disposizione diversi tipi di collezioni.

### 5.10.1 TArray

E' un array dinamico di oggetti.

Metodi utili:

- `Add(element)`: aggiunge element alla lista
- `AddUnique(element)`: aggiunge element alla lista solo se element non e' gia' presente nella lista (verifica con operatore `==`)
- `Insert(element, index)`: inserisce element nella lista in posizione index
- `Sort()`: esegue l'ordinamento della lista. Gli oggetti devono essere confrontabili con l'operatore di minore "`<`".
- `Num()`: restituisce il numero di oggetti nella lista
- `Find(element)`: restituisce l'indice di element oppure "INDEX\_NONE" (-1) se l'elemento non e' nella lista.
- `Remove(element)`: rimuove tutte le occorrenze di element dalla lista
- `RemoveSingle(element)`: rimuove solo la prima occorrenza di element dalla lista

### 5.10.2 TMap

E' una mappa/dizionario: memorizza coppie chiave-valore.

Esempio di dichiarazione con valori:

```
// i tipi devono corrispondere
TMap<int, FString> MiaMappa = {
    {0, "A"},
    {1, "B"},
    // ...
};
```

Metodi utili:

- `Add(element)`: aggiunge element alla lista
- `Contains(key)`: restituisce true se la chiave esiste nella mappa
- `Find(key)`: restituisce un puntatore al valore corrispondente alla chiave oppure nullptr se la mappa non contiene quella chiave.
- `FindRef(key)`: restituisce una copia del valore corrispondente alla chiave oppure un oggetto con il valore di "default" dello stesso tipo del valore atteso.

## 5.11 Enumerazioni

Le enumerazioni permettono di creare un tipo personalizzato specificando tutti i suoi possibili valori.

Una variabile di tipo enumerazione deve assumere obbligatoriamente un valore contenuto tra l'insieme dei valori specificati.

Per definire una enumerazione bisogna creare un header file con il seguente contenuto:

```
#pragma once

UENUM()
enum ENomeDellaEnumerazione {
    // elementi separati da virgola
};
```

Modificare in modo opportuno "NomeDellaEnumerazione" con il nome che si vuole dare all'enumerazione e specificare tutti i possibili valori dell'enumerazione separati da virgola all'interno delle parentesi graffe.

E' possibile avere un attributo del tipo dell'enumerazione aggiungendolo in questo modo:

```
UPROPERTY()
TEnumAsByte<ENomeDellaEnumerazione> NomeAttributo;
```

## 5.12 Funzioni e macro utili

Questa sezione contiene funzioni e macro utili che non appartengono alle classi specificate in questo capitolo.

- Cast<T>(): permette di eseguire in modo safe il casting di un puntatore.

Esempio d'uso:

```
// VecchiaVariabile e' un puntatore a VecchioTipo.
// Voglio eseguire il casting per renderlo puntatore a NuovoTipo
NuovoTipo* NuovaVariabile = Cast<NuovoTipo>(VecchiaVariabile);
```

- UKismetMathLibrary::IsPointInBox(): permette di verificare se una coordinata e' contenuta all'interno di un volume a forma di cubo.

Per utilizzare il metodo e' necessario includere l'header:

```
#include "Kismet/KismetMathLibrary.h"
```

Esempio d'uso:

```
UKismetMathLibrary::IsPointInBox(
    Pallet->GetActorLocation(),

    // utilizzo le coordinate di questo oggetto (this) e le sue dimensioni
    // per verificare se l'actor Pallet e' contenuto in questo oggetto
    this->GetActorLocation(),
    this->GetComponentsBoundingBox().GetExtent()
)
```

- UGameplayStatics::GetAllActorsOfClass(): permette di recuperare una lista di tutti gli actor di una classe.

```
#include "Runtime/Engine/Classes/Kismet/GameplayStatics.h"
```

```
TArray<AActor*> FoundActors;
```

```
UGameplayStatics::GetAllActorsOfClass(
    // mondo che contiene gli actor
    GetWorld(),
    // classe degli actor
    AClasseDelActor::StaticClass(),
    // array che conterrà gli actor trovati
    FoundActors
);
```

**NOTA**

Per utilizzare i metodi degli actor trovati è necessario eseguire il casting dei puntatori nel TArray poiché il metodo popola il TArray con puntatori ad AActor generici.

- FMath::VInterpConstantTo(): dato un punto di partenza, un punto di arrivo, DeltaTime (frame) e velocità calcola il prossimo punto in cui si dovrebbe trovare un oggetto nel prossimo frame.

```
FMath::VInterpConstantTo(
    // coordinate attuali
    this->CurrentLocation,
    // coordinate di destinazione
    this->TargetLocation,
    // frame, parametro di Tick()
    DeltaTime,
    // velocità
    this->InterpSpeed)
```

### 5.13 Delegate e eventi custom

I delegate permettono di creare eventi, legare metodi agli eventi per eseguirli quando l'evento viene lanciato, lanciare un evento e iscriversi all'evento.

Esistono diversi tipi di delegate:

- Single: eseguono una singola callback
- Multicast: possono eseguire più di una callback
- Events: come i Multicast ma l'unica classe che può eseguire le callback è la classe che ha dichiarato gli eventi.
- Dynamic "Single": come Single ma le funzioni possono essere recuperate specificandone il nome
- Dynamic "Multicast": come Multicast ma le funzioni possono essere recuperate specificandone il nome

**NOTA**

Per maggiori informazioni fare riferimento alla documentazione ufficiale

I seguenti esempi saranno relativi ai delegate "Single" ma l'uso degli altri delegate è praticamente equivalente.

Per prima cosa devono essere dichiarati:

```
// dichiara un delegate di nome DelegateName.
// la callback è void senza parametri
DECLARE_DELEGATE(DelegateName)
```

```

// dichiara un delegate di nome DelegateName.
// la callback e' void e ha <Num> parametri dei tipi elencati.
// > <Num> appartiene all'insieme {One, Two, Three, Four, Five, ecc...}
DECLARE_DELEGATE_<Num>Params(DelegateName, Param1Type, Param2Type, ...)

// dichiara un delegate di nome DelegateName.
// la callback restituisce una variabile di tipo RetValType e non ha parametri
DECLARE_DELEGATE_RetVal(RetValType, DelegateName)

// dichiara un delegate di nome DelegateName.
// la callback restituisce una variabile di tipo RetValType
// e ha <Num> parametri dei tipi elencati.
// > <Num> appartiene all'insieme {One, Two, Three, Four, Five, ecc...}
DECLARE_DELEGATE_RetVal_<Num>Params(RetValType, DelegateName, Param1Type, Param2Type, ...)

```

Per utilizzare il delegate bisogna definire una variabile che la contiene.

```

// dichiariamo fuori dalla classe il delegate:
DECLARE_DELEGATE(DelegateName)

// classe che invierà le notifiche
UPROPERTY()
DelegateName MioDelegate;

// posso usare un metodo setter a cui passo un reference al delegate
// per poter permettere dall'esterno di iscriversi al delegate
void SetMioDelegate(DelegateName& InMioDelegate)
{
    this->MioDelegate = InMioDelegate;
}

```

Poi deve essere eseguito il bind tra le callback che devono essere chiamate e i delegate stessi:

```

// classe "AMiaClasseRicevitore" che deve ricevere le notifiche
UPROPERTY()
DelegateName MioDelegate;

// esegui il bind: verrà chiamata LaMiaCallback
void CreateBinding()
{
    MioDelegate.Bind(
        // oggetto che contiene la callback da richiamare.
        // puo' essere anche un altro oggetto di un'altra classe
        this,
        // AMiaClasseRicevitore puo' essere una classe diversa
        // da quella di this (deve avere il metodo LaMiaCallback)
&AMiaClasseRicevitore::LaMiaCallback
    );
}

void LaMiaCallback()
{
    UE_LOG(LogTemp, Warning, TEXT("Hai chiamato la callback"));
}

```

Infine l'oggetto che deve eseguire le callback deve richiamare il metodo:

```
if (MioDelegate.IsBound())
{
    // assicurati che ci sia un qualche
    // legame prima di tentare l'esecuzione
    MioDelegate.Execute();
}
```

## 5.14 Timer

I timer permettono di eseguire un metodo dopo una certa quantita' di tempo e, eventualmente, di ripetere l'esecuzione del metodo ogni x secondi.

```
// imposta attraverso il gestore di timer un nuovo timer
GetWorldTimerManager().SetTimer(
    // oggetto per gestire il nuovo timer
    // > definizione: struct FTimerHandle WaitTimerHandle;
    WaitTimerHandle,
    // oggetto che ha il metodo da eseguire
    this,
    // metodo da eseguire
    &AMiaClasse::LaMiaCallback,
    // intervallo (in secondi) ogni quanto eseguire il metodo
    // > se il parametro successivo e' false questo valore viene ignorato
    2.0f,
    // false = esegui il metodo una volta sola,
    // true = esegui il metodo ogni intervallo di tempo
    false,
    // tempo (in secondi) atteso prima di eseguire per la prima volta il metodo
    this->TimeToWait
);
```

## 5.15 FString

FString e' una stringa mutabile di dimensione variabile.

Metodi utili:

- IsEmpty(): restituisce true se la stringa e' vuota.
- Equals(): verifica se la stringa e' uguale ad un'altra stringa.
- Split(): separa la stringa su cui viene chiamato il metodo in due parti "Left" e "Right" su un carattere.

### ESEMPIO

```
// ottieni MyString da qualche parte
FString MyString = FString(TEXT("This is my test FString."));

// separa MyString in due parti,
// usa <Separator> come carattere separatore
FString Left, Right;
MyString.Split(TEXT("<Separator>"), &Left, &Right);
```

# Capitolo 6

## Plugin

I plugin aggiungono funzionalita' aggiuntive ad Unreal Engine.

Esistono plugin:

- già inclusi in Unreal Engine: non bisogna installarli
- da installare mediante interfaccia di Unreal Engine andando su:

`Edit > Plugins`

Verra' aperta la schermata nella figura 6.1.

- da installare scaricandoli, ad esempio da GitHub, e posizionandoli nella cartella "Plugins" del progetto

### 6.1 Geometry Script

Permette di creare delle mesh dinamiche semplici.

### 6.2 Datasmith

I plugin Datasmith permettono di importare in Unreal Engine formati che non sono supportati di default.

#### 6.2.1 Datasmith CAD importer

Permette di importare file AutoCAD.

### 6.3 MQTT Utilities

"MQTT Utilities" è un plugin scaricabile da GitHub che aggiunge ad Unreal Engine la possibilità di gestire il protocollo MQTT (inviare e ricevere messaggi dal broker).

#### ATTENZIONE

L'attuale ultima versione, ovvero la v1.1.0, ha un bug che fa crashare l'engine.

Se non sono presenti nuove release è consigliato scaricare il plugin direttamente dal branch master: il codice del branch master ha il bug fix che risolve il problema.

Per installare il plugin occorre:

1. Scaricarlo
2. Creare, se non esiste, una cartella "Plugins" all'interno della cartella del progetto Unreal Engine

3. Creare una cartella per il plugin chiamata "mqtt-utilities-unreal" e spostare i file della root del plugin al suo interno.

#### NOTA

Ad esempio se si sta lavorando su un progetto chiamato "DigitalShadowLED" navigare nella sua cartella (quella che contiene il file ".uproject").

Il file README dovrà essere posizionato all'interno della cartella:

`DigitalShadowLED\Plugins\mqtt-utilities-unreal`

Dove "DigitalShadowLED" è la cartella contenente il file ".uproject".

4. Aprire Unreal Engine

5. Aprire la lista dei plugin (figura 6.1) e abilitare il plugin da:

`Edit > Plugins > Code Plugins`

6. Aprire il file che termina con "Build.cs" (presente nella sottocartella contenuta nella cartella "Source") e modificare la seguente riga in modo da contenere "MqttUtilities":

```
PublicDependencyModuleNames.AddRange(new string[] { "Core", ... , "MqttUtilities" });
```

Figura 6.1: Unreal Engine: Plugin

Plugin	Description	Version	Provider
Ability System Game Feature Actions	Ability System Game Feature Actions	Version 0.1	Epic Games, Inc.
Actor Layer Utilities	Utilities for interacting with actor layers from blueprints	Version 1.0	
Actor Palette (Beta)	Allows creation of Actor Palettes based on existing levels to quickly select actors and drag them into the level editor	Version 1.0	Epic Games, Inc.
Actor Sequence (Experimental) (Beta)	Runtime for embedded actor sequences	Version 0.1	Epic Games, Inc.
Adjust Analytics Provider	Adjust Analytics Provider	Version 1.0	Epic Games, Inc.
ADO Support	ADO (ActiveX Data Objects) Database Support	Version 1.0	Epic Games, Inc.
AES GCM network packet handler (Beta)	Provides a packet handler component to do AES GCM encryption and decryption.	Version 1.0	Epic Games, Inc.
AES network packet handler	Provides a packet handler Component to do AES encryption and decryption.	Version 1.0	Epic Games, Inc.

# Capitolo 7

## Annotazioni

### 7.1 Asset

Per trovare gli asset e' comodo cercare nel market place di Unreal Engine oppure su Quixel.

SketchFab contiene degli asset gratuiti e a pagamento.

Un altro sito che contiene invece personaggi e animazioni per i personaggi e' Mixamo.

### 7.2 Environment Light Mixer

Strumento che permette di lavorare con la luce.

### 7.3 Hotkey utili

- shift + F1 permette di ottenere il controllo del mouse mentre sta andando il gioco.
- tab + ESC permette di fermare il gioco che e' in esecuzione.
- alt + P fa partire il gioco.



# Capitolo 8

## Troubleshooting

### NOTA

Questa sezione e' stata realizzata per risolvere problemi con Unreal Engine 5.0.3 e Visual Studio 2022 Community Edition su sistema operativo Windows 10.

### 8.1 Visual Studio

Se Visual Studio non trova header che esistono oppure si desidera eliminare dei file dal progetto puo' essere utile far ricreare da Unreal Engine i file per Visual Studio: fare click con il tasto destro sul file .uproject e selezionare "Generate Visual Studio project files".

### ATTENZIONE

Per generare i file e' necessario installare ".NET Runtime 3.1.30".

### CONSIGLIO

Potrebbe essere utile cancellare le cartelle "Binaries", "DerivedDataCache", "Intermediate", "Saved" e il file ".sln" prima di generare i file.

### 8.2 Packaging

I seguenti errori sono relativi al packaging del gioco.

#### 8.2.1 Errore: the sdk for windows is not installed properly

Questo errore appare quando non si e' installato ".NET Desktop Runtime 3.1.30" per Windows.

### 8.3 Evitare flickering delle ombre

Per evitare il flickering delle ombre bisogna andare in:

`Settings > project settings > engine > rendering`

E impostare l'opzione "Shadow Map Method" a "Shadow Maps".  
Un'altra opzione e' impostare la mobility delle luci a "Movable".

## 8.4 Crash di Unreal Engine

Se Unreal Engine crasha e' molto probabile che si stia cercando di accedere ad un puntatore nullo o invalido.

Alcune possibili cause sono:

- Non specificare sopra ad un TArray di puntatori "UPROPERTY()": il garbage collector ha eliminato per errore oggetti che non dovevano essere eliminati.

Aggiungere "UPROPERTY()" permette di dire ad Unreal Engine di gestire la memoria dinamica in modo corretto.

Se non si aggiunge UPROPERTY si rischia di accedere ad un puntatore che non e' piu' valido facendo crashare Unreal Engine.

- Utilizzare Find() su una TMap ed accedere immediatamente al puntatore restituito: Find() restituisce un puntatore nullptr se non e' stato possibile trovare un oggetto con la chiave specificata.

Accedere al puntatore nullo fa crashare il motore grafico.

- Utilizzare NewObject all'interno di un costruttore: NewObject restituira' un puntatore nullptr che, una volta acceduto, fara' crashare l'editor.

Per risolvere il problema richiamare NewObject in un metodo diverso come, ad esempio, il metodo BeginPlay.

## 8.5 Errori di compilazione

### 8.5.1 Nested Containers are not supported

Questo errore indica che si sta usando UPROPERTY() su un attributo che e' un TArray che contiene un TArray, una TMap che contiene una TMap, ... E tutte le varie combinazioni di collezioni innestate una dentro l'altra.

Unreal Engine non supporta la dichiarazione di queste variabili come UPROPERTY().

Per risolvere il problema e' possibile creare una classe che eredita dal tipo interno e poi creare una collezione di istanze della classe creata.

#### NOTA

Esempio:

```
UPROPERTY()
TArray<TArray<int>> mioAttributo;
```

Questo codice fara' lanciare un errore di compilazione.

Si potrebbe pensare di risolvere il problema rimuovendo UPROPERTY():

```
TArray<TArray<int>> mioAttributo;
```

Questa soluzione compila ma senza UPROPERTY si rischia che il garbage collector elimini il contenuto dell'array (con "int" NON DOVREBBE succedere ma con oggetti o con puntatori ad oggetti piu' complessi si).

La soluzione e' creare una classe MioContenitore che eredita da TArray<int>. In questo modo l'attributo sara' di tipo:

```
UPROPERTY()
TArray<MioContenitore> mioAttributo;
```

Questa soluzione compila e il contenuto del TArray non sara' eliminato erroneamente dal garbage collector.

Un'altra soluzione e' creare una collezione di struct/oggetti e ogni struct/oggetto contiene un attributo con la collezione interna.



# Capitolo 9

## Bibliografia

- [1] Unreal Engine Community. *Unreal Engine Community Wiki*. <https://unrealcommunity.wiki/>. Accessed: 2023-04-03.
- [2] Lötwig Fusel. *UnrealEngine 5 / C++ Tutorial*. <https://www.youtube.com/playlist?list=PL-m4pn2uJvXHL5rxdukhqrSRM5gN43YN>. Accessed: 2022-11-01.
- [3] Epic Games. *Your First Hour in Unreal Engine 5*. <https://dev.epicgames.com/community/learning/courses/ZpX/your-first-hour-in-unreal-engine-5/RPwK/your-first-hour-in-unreal-engine-5-overview>. Accessed: 2022-11-01.
- [4] Harrison1. *unrealcpp*. <https://github.com/Harrison1/unrealcpp>. Accessed: 2022-11-01.
- [5] insthync. *awesome-unreal*. <https://github.com/insthync/awesome-unreal>. Accessed: 2022-11-01.
- [6] *Learn C++*. <https://www.learn.cpp.com/>. Accessed: 2022-11-01.
- [7] Unreal Sensei. *UE5 Starter Course 2022*. <https://www.youtube.com/watch?v=k-zAkzmduqI>. Accessed: 2022-11-01.
- [8] *Unreal Engine 5 Documentation*. <https://docs.unrealengine.com/5.0/en-US/>. Accessed: 2022-11-01.