Model Configurator

Serie di componenti per la gestione della personalizzazione di un oggetto. In particolare è stato pensato per la configurazione di un auto con l’utilizzo di altri componenti nel pacchetto BUNNY\_TK, ma è anche applicabile ad altro.

Contenuti:

* Scripts, component/classi principali
* Tests, scene e componenti di test
* Samples, esempi di estensione di componenti
* Editor, custom inspectors

Definizioni:

* **Definizione**: tutte le possibili personalizzazioni sono definite all’interno di una definizione (*ConfigurationDefinitions*). Le personalizzazioni sono raggruppati per *tipi* e all’interno di una definizione sono descritte tutte i possibili *valori* di un tipo (es: colore [rosso, blu, nero]), da notare che tutti i valori di un tipo sono esclusivi tra loro (es. il colore non può essere rosso e blu).
* **Configurazione**: data una definizione, una configurazione specifica per ogni tipo il valore. In particolare i tipi possono assumere valore *non definito* (*UNDEFINED*) ed in tal caso viene ignorato nelle operazioni di confronto e di overlap. Es: colore:rosso, ruote:A, spoiler:undefined (qualiasi).
* **Overlap di configurazioni**: due configurazioni si possono *sommare* per ottenere una configurazione che racchiuda entrambi. Se entrambi le configurazioni modificano lo stesso tipo, tale tipo assumerà quella della seconda configurazione.
* **Restrizioni**: una restrizione è una intersezione di 2 insieme di valori, appartenente ognuno allo stesso tipo, che non possono essere applicati contemporaneamente al modello (es. colore:rosso, blu e ruote:A significa che l’auto non può avere le ruote A se è di colore rosso o blu). Inoltre in una restrizione viene definito il valore da assumere in caso di conflitto (*fallback*).

Note Generali

Struttura

* **ConfigValue**, *l’unità* più piccola della struttura, definisce il tipo e il valore (es. colore:rosso), e ha le funzionalità di confronto e di overlap. Tutta la struttura lavora con interi/indici, sono poi semplificati e resi leggibili tramite inspector personalizzati.
* **ConfigurationDefinitions**, oltre alle definizioni dei tipi e possibili valori ha anche le restrizioni e le funzionalità di overlap con appunto l’applicazione delle restrizioni. Da notare che questa componente viene utilizzato principalmente per le restrizioni e per le funzioni di leggibilità e facilità d’uso delle configurazioni (inspector personalizzati).
* **ConfigMaskMatch**, rappresenta una restrizione, quindi riferimenti agli insiemi di valori e di valori di fallback.
* **ConfigurationIDBase** classe base che serve per identificare una **Configuration**. Una specializzazione di questa classe deve implementare le operazioni di Similar, Same e Overlap e le funzioni per ottenere i valori in *string*, indispensabili per la gestione delle configurazioni.
* **ConfigurationID**, è una specializzazione della classe **ConfigurationIDBase**,strutturato in maniera *generica* in modo da poter essere configurato velocemente.Racchiude un insieme di più **ConfigValue** per creare una configurazione, ha il riferimento alla definizione e le funzionalità di overlap e di confronto. Affinché le operazioni di confronto e di Overlap avvengano correttamente i ConfigurationID che fanno riferimento ad una stessa definizione deve avere N ConfigValue quanti sono i tipi possibili definiti nella ConfigurationDefinitions.
* **Configuration**, ha le funzionalità di applicazione tramite MaterialManager e GameObjectGroup sul modello (e di rimozione se possibile) di una configurazione. Necessità di un ConfigurationID per essere identificato dal ModelConfiguratorManager.
* **MaterialManager**, ha la funzionalità di applicare un *materiale* ad una lista di *mesh*.
* **GameObjectGroup**, è semplicemente un raggruppamento di *gameObject* con la funzionalità di attivazione/disattivazione en masse.
* **ModelConfiguratorBase**/**ModelConfiguratorManager**, contiene i riferimenti di tutte le possibili configurazioni e gestisce lo stato del modello.

Overlap

In seguito un esempio per spiegare il funzionamento dell’overlap. La funzionalità è definita all’interno di ConfigValue e viene richiamato dalla ConfigurationID alla quale *appartiene*. In particolare nella tabella sono evidenziate i campi che *determinano* il risultato.

*configurazioneRisultato* = *configurazioneA*.**Overlap**(*configurazioneB*)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OVERLAP | | | | |
| # | TIPI | **CONFIGURAZIONE RISULTATO** | **CONFIGURAZIONE A** | **CONFIGURAZIONE B** |
| 1. | *Colore* | Rosso | Rosso | Undefined |
| 2. | *Cerchio* | 21 Offroad | Undefined | 21” Offroad |
| 3. | *Spoiler* | None | Sport | None |
| 4. | *Interni* | Undefined | Undefined | Undefined |

Essenzialmente si riassume in:

* Se uno dei 2 valori è UNDEFINED, il risultato sarà uguale a quello non nullo (vedi riga 1 e riga 2).
* Se entrambi non sono UNDEFINED, il risultato sarà uguale a quello del secondo (vedi riga 3).
* Se entrambi sono UNDEFINED, il risultato sarà UNDEFINED (vedi riga 4).

Note:

* Il valore UNDEFINED è *unico* per tutti i tipi, esso è definito nella classe ConfigValue con valore -1.
* Ovviamente 2 ConfigValue devono essere dello stesso tipo affinché l’overlap sia possibile.
* La funzione Overlap in ConfigurationID effettua prima operazioni di *omogeneizzazione* affinché i suoi ConfigValue siano allineati (per numero e per tipo) con l’altra ConfigurationID.
* Il risultato della funzione Overlap in ModelConfiguratorManager viene salvato in currentConfiguration.

Operazioni di confronto

In seguito un esempio di funzionamento delle funzioni Same e Similar tra due configurazioni. Tali funzionalità sono definiti all’interno di ConfigValue che viene semplicemente richiamato dalla ConfigurationID alla quale *appartiene*. In particolare nella tabella sono evidenziati i valori per cui le due configurazioni non sono simili o uguali.

*configurazioneA*.**Similar**(*configurazioneB*)

*configurazioneA*.**Same**(*configurazioneB*)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **CONFRONTI** | | | |
| **#** | **SAME** | **SIMILAR** | **CONFIGURAZIONE A** | **CONFIGURAZIONE B** |
| 1 | FALSE | FALSE | Rosso | Undefined |
| 2 | Undefined | 21” Offroad |
| 3 | Sport | None |
| 4 | Undefined | Undefined |
| 5 | FALSE | TRUE | Rosso | Undefined |
| 6 | Undefined | 21” Offroad |
| 7 | Undefined | None |
| 8 | Undefined | Undefined |
| 9 | TRUE | TRUE | Rosso | Rosso |
| 10 | 21” Offroad | 21” Offroad |
| 11 | Sport | Sport |
| 12 | Undefined | Undefined |

Essenzialmente si riassume in:

* 2 configurazioni sono SAME (uguali) se tutti i valori hanno lo stesso valore (vedi riga 9 – 12)
* 2 configurazioni sono SIMILAR (simili) se tutti i valori sono uguali o l'altro è Undefined (vedi riga 5 – 8).

Note:

* Come per l’operazione di Overlap, il valore UNDEFINED è unico per tutti i tipi.
* Nel caso dell’operazione Similar, le strutture di entrambi le configurazioni possono essere disallineate.
* Nel caso dell’operazione Same, le strutture di entrambi le configurazioni devono essere allineate.
* Se due configurazioni non sono simili non possono essere uguali.

Istruzioni di utilizzo

Inizializzazione della struttura di configurazione

Per utilizzare i componenti basta importare il pacchetto *BUNNY\_TK.package* nel progetto di destinazione, in particolare le cartelle interessate sono *ModelConfigurator* e *General*.

Una volta importato creare la definizione da utilizzare:

1. Nella finestra *Project* creare una cartella di nome *Resources* in qualunque percorso negli *Assets*.
2. All’interno della cartella *Resources* cliccare col destro e selezionare quindi *Create* → *Utilities* → *ConfigurationDefinitions* e dare quindi il nome all’oggetto.

Selezionando la definizione dalla finestra *Project*, sarà possibile modificarlo nella finestra *Inspector* utilizzando i relativi bottoni, quindi aggiungere i tipi e i relativi valori e le restrizioni se necessario.

Ora non resta altro che inizializzare il *manager* e le relative configurazioni nella scena.

1. Nella scena di destinazione creare un oggetto vuoto che ospiterà il componente *manager* ed aggiungere il componente ConfigurationID.
2. Aggiungere quindi il componente *ModelConfiguratorManager* all’oggetto, assicurarsi che nel campo *Definitions* sia selezionato la definizione
3. Comparirà un bottone *Initialize* sul componente *ModelConfiguratorManager*, cliccandolo verranno create:
   1. N figli per ogni tipo, senza nessun componente, con la semplice funzione di raggruppamento
   2. Ogni “figlio-tipo” avrà M figli per quanti sono i possibili valori di quel tipo. Quest’ultimi avranno ognuno un componente *ConfigurationID* già inizializzato ed un componente *Configuration* (ancora da configurare).

Note importanti:

* E’ importante che la definizione sia in una cartella di nome *Resources* in modo che lo si possa caricare in esecuzione. Non importa dove la cartella *Resources* venga creata, l’importante che sia in una cartella all’interno di *Assets* accessibile dall’applicazione ([documentazione](https://docs.unity3d.com/ScriptReference/Resources.html)).
* Il bottone *Initialize* del componente *ModelConfiguratorManager* non tiene conto degli oggetti già esistenti sotto la sua gerarchia né tiene traccia di quelli che sono stati creati, pertanto in caso di un nuovo *Initialize* è consigliabile cancellare manualmente gli oggetti precedentemente creati.
* Le azioni del bottone *Initialize* non sono registrati dalle funzioni di *Annulla* (*Undo, CTRL+Z)*.
* Da notare che *ModelConfiguratorManager* deve avere i riferimenti dei *ConfigurationID*, quindi in caso di cancellazioni o creazioni di nuovi, assicurarsi di aggiornare tali riferimenti utilizzando il bottone *Update*.
* Le condizioni di ricerca del bottone *Update* sono configurabili; per ricerca nella gerarchia (*In Children*) o nell’intera scena (*Global*) e se devono avere la stessa definizione selezionata nel campo *Definitions*.

A questo punto la struttura per la gestione delle configurazioni è inizializzato. Resta da sistemare l’applicazione *grafica/visuale* della configurazione al modello.

Configurations, MaterialManager e GameObjectGroup

Affinché la comunicazione tra modello e *ModelConfiguratorManager* avvenga vengono utilizzati i seguenti componenti:

* **Configuration** è il componente che si interfaccia con il *manager*.
* **MaterialManager** si occupa di applicare un materiale ad una serie di *mesh*.
* **GameObjectGroup** raggruppa vari gameObject (es. ruote, fari) che devono essere attivati/disattivati.

Configuration

Ogni componente *Configuration* deve avere:

* Un identificativo che deriva da *ConfigurationIDBase*, se è stato utilizzato il bottone *Initialize* di *ModelConfiguratorManager* un componente identificativo (*ConfigurationID*) è stato aggiunto automaticamente.
* Riferimenti ai *GameObjectGroup* che deve attivare/disattivare quando viene applicato.
* Riferimenti ai *MaterialManager* che deve applicare.

GameObjectGroup

Campi:

* **GameObjects** è la lista di gameObject influenzati da questo componente.
* **Is Active** è lo stato del gruppo.
* **Game Objects To Disable** sono gli i gameObject che vengono disattivati quando questo component **IsActive** è vero e riattivati quando **IsActive** è falso.
* **Groups To Disable** funziona come *GameObjectsToDisable* solo che fa riferimenti ad altri *GameObjectGroup*.

Ci sono inoltre alcune funzionalità nell’*Editor*:

* **Select target GOs** che seleziona i gameObject influenzati da questo componente, utile per avere una visualizzazione più chiara nella finestra *Scene*.
* **Select GOs to disable** che seleziona i gameObject nella lista *GameObjectsToDisable* (quelli da disattivare)
* **Toggle** inverte lo stato, essenzialmente un’anteprima degli stati di questo componente.

Note importanti:

* Per aggiungere gameObjects basta trascinarli nel campo desiderato.
* I Configurations sono esclusivi tra loro. Quando *ModelConfiguratorManager* effettua un *Apply* viene prima chiamato *Remove* su tutte le configurazioni e poi *Apply* a tutte le configurazioni simili, di conseguenza, una disattivazione dei GameObjectGroup.

MaterialManager

Campi:

* **Name** nome, non necessario.
* **Target Meshes** è la lista di MeshRenderer alla quale verrà applicato il materiale.
* **Target Material** è il materiale da applicare alle mesh.
* **Index Material** è l’indice del materiale da sostituire, utilizzato solo nel caso in cui le mesh sono multi-materiale.

Ci sono inoltre alcune funzionalità nell’*Editor*:

* **Apply Material** applica il materiale alle mesh.
* **Select Meshes** seleziona tutte le mesh influenzate, in modo da avere una visualizzazione più chiara nella finestra *Scene*.
* **Add/Remove Target** è il target alla quale verranno eseguiti le operazioni dai bottoni sottostanti.
* **Add Target Mesh** semplicemente aggiungerà alla lista delle mesh, il MeshRenderer del Target se necessario.
* **Add Target Meshes Child** aggiunge tutti i MeshRenderer sotto la gerarchia (figli di) Target se non sono già presenti nella lista delle mesh.
* **Add Meshses from MaterialManager** se Target ha un componente *MaterialManager* aggiungerà la lista di quest’ultimo alla propria, utile per copiare.
* **Remove Target Mesh** rimuove il MeshRenderer di Target dalla lista.
* **Remove Target Meshes Child** rimuove tutti i MeshRenderer sotto la gerarchia (figli di) Target dalla lista delle mesh.
* **Remove Meshes Selected** rimuove tutti i MeshRenderer attualmente selezionate nella finestra *Hierarchy.*
* **Toggle Active GOs**inverte lo stato dei gameObject nella lista delle mesh.

Note importanti:

* L’operazione di *ApplyMaterial* non è registrato dalla funzione *Annulla* (*Undo*).
* E’ possibile aggiungere/rimuovere mesh senza utilizzare i bottoni di *Add/Remove* trascinando o eliminandoli (click destro su elemento) dalla lista *TargetMeshes*.

Cosa avviene quando…

Quando viene applicato una configurazione avvengono i seguenti passaggi:

1. Chiamata a ModelConfiguratorManager di ApplyConfiguration per una certa configurazione.
2. ModelConfiguratorManager effettua l’overlap tra la configurazione attuale e la configurazione da applicare.
3. La chiamata di overlap è passata al ConfigurationID attuale che lo passa a sua volta ai suoi ConfigValue, infine, se il riferimento alla definizione non è nullo viene chiamato anche l’applicazione delle restrizioni.
4. Una volta ottenuto il risultato dell’overlap diventa quindi quello attuale. ModelConfiguratorManager chiama la funzione di *Apply* su tutte le configurazioni *simili* alla configurazione attuale e *Remove* a quelli non.
5. Configuration, infine, inoltra le chiamate *Apply* e *Remove* ai suoi *MaterialManager* e *GameObjectGroup* che si occupano di applicare/disattivare materiali/gameObjects.