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**/**ConfigurationID**, 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.
* **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

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 nella finestra *Inspector* sarà possibile modificarlo utilizzando i relativi bottoni, quindi aggiungere i tipi e i relativi valori.

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. Trascinare nel campo *Definitions* l’oggetto definizione creato nella cartella *Resources.*
3. Aggiungere quindi il componente *ModelConfiguratorManager* all’oggetto, anche qui nel campo *Definitions* trascinare l’oggetto definizione all’interno della cartella *Resources*.
4. Comparirà un bottone *Initialize* sul componente *ModelConfiguratorManager*, cliccandolo verranno create le configurazioni raggruppate per tipo nella gerarchia.

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.