Don't Panic

3DMob: Grafica 3D su device mobili

Definizione di Prodotto

Informazioni sul documento

Versione	5.2.0
Redazione	Rampazzo Federico Basaglia Mattia
Verifica	Pezzutti Marco Lain Daniele
Responsabile	Cesarato Fabio
Uso	Esterno
Lista di distribuzione	Don't Panic Prof. Vardanega Tullio Prof. Cardin Riccardo

Descrizione

Architettura di dettaglio dell'applicazione 3DMob

Diario delle modifiche

Descrizione modifica	Autore	Ruolo	Data	Versione
Approvazione documento	Cesarato Fabio	Responsabile	2013-03-11	5.2.0
Verifica documento	Pezzutti Marco	Verificatore	2013-03-9	5.1.1
Verifica documento	Lain Daniele	Verificatore	2013-03-9	5.1.0
Aggiunti i parametri ai metodi	Rampazzo Federico	Progettista	2013-03-8	5.0.6
Aggiunta appendice sui comandi	Rampazzo Federico	Progettista	2013-03-7	5.0.5
Aggiunte descrizioni diagrammi sequenza	Basaglia Mattia	Progettista	2013-03-6	5.0.4
Aggiornati XMLSchema e JSONSchema	Basaglia Mattia	Progettista	2013-03-6	5.0.3
Aggiunti parametri ai metodi	Rampazzo Federico	Progettista	2013-03-5	5.0.2
Ristrutturazione del documento	Rampazzo Federico	Progettista	2013-03-5	5.0.1
Approvazione documento	Basaglia Mattia	Responsabile	2013-02-17	4.2.0
Verifica documento	Pezzutti Marco	Verificatore	2013-02-16	4.1.1
Verifica documento	Lain Daniele	Verificatore	2013-02-15	4.1.0
Appendice schemi file esportati	Sciarrone Riccardo	Progettista	2013-02-12	4.0.4
Tracciamento	Cesarato Fabio	Progettista	2013-02-10	4.0.3
Specifica componenti	Rampazzo Federico	Progettista	2013-02-09	4.0.2
Stesura standard di progetto	Busato Luca	Progettista	2013-02-08	4.0.1
Creazione scheletro del documento e stesura introduzione	Cesarato Fabio	Progettista	2013-02-07	4.0.0

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il seguente documento ha lo scopo di definire nel dettaglio la struttura del sistema 3DMob, approfondendo quanto già riportato nella *Specifica Tecnica*. Tale documento fornisce una struttura dettagliata e completa che viene utilizzata dai *programmatori* per le attività di codifica.

1.2 Scopo del Prodotto

Lo scopo del progetto è la realizzazione di un applicazione in grado di convertire file prodotti da programmi di grafica 3D in file in formato JSON$_G$ in grado di essere visualizzati su dispositivi mobile senza perdita di informazione. L'obiettivo è quello di semplificare il workflow attuale necessario a rendere compatibili i file.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento *Glossario v5.2.0*.
Ogni occorrenza di vocaboli presenti nel *Glossario* è marcata da una "G" maiuscola in pedice.

1.4 Riferimenti

1.4.1 Normativi

- **Specifica Tecnica**: *Specifica Tecnica v4.2.0*;
- **Analisi dei Requisiti**: *Analisi dei Requisiti v4.2.0*;
- **Norme di Progetto**: *Norme di Progetto v5.2.0*.

1.4.2 Informativi

- **Documentazione Qt$_G$ per Segnali e Slot**
  http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html;
- **Documentazione di Assimp**
  http://assimp.sourceforge.net/lib_html/index.html;
- **Documentazione Boost.PropertyTree**
  http://www.boost.org/doc/libs/1_52_0/doc/html/property_tree.html;
- **Documentazione Qt$_G$'s Undo Framework$_G$**
  http://qt-project.org/doc/qt-5.0/qtdoc/qundo.html;
- **Glossario**: *Glossario v5.2.0*.

2 Standard di progetto

2.1 Standard di progettazione architetturale

Gli standard di progettazione architetturale sono definiti nella *Specifica Tecnica v4.2.0*.

2.2 Standard di documentazione del codice

Gli standard per la scrittura della documentazione del codice sono definiti nelle *Norme di Progetto v5.2.0*.

2.3 Standard di denominazione di entità e relazioni

Tutti gli elementi definiti, siano essi package$_G$, classi, metodi o attributi, devono avere denominazioni chiare ed autoesplicative. Nel caso in cui il nome risulti essere lungo è preferibile anteporre la chiarezza alla lunghezza.
Sono ammesse abbreviazioni se:

- immediatamente comprensibili;
- non ambigue;
- sufficientemente contestualizzate.

Le regole tipografiche relativi ai nomi delle entità sono definiti nelle *Norme di Progetto v5.2.0*.

2.4 Standard di programmazione

Gli standard di programmazione sono definiti e descritti nelle *Norme di Progetto v5.2.0*.

2.5 Strumenti di lavoro

Gli strumenti da adottare e le procedure da seguire per utilizzarli correttamente durante la realizzazione del prodotto software sono definiti nelle *Norme di Progetto v5.2.0*.

3 Specifica componenti

3.1 DDDMob

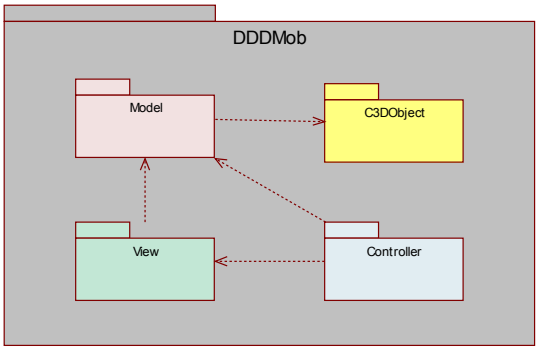

Figura 1: Componente DDDMob

Namespace$_G$ globale per il progetto. Le relazioni tra i package$_G$ Model, View e Controller identificano le relazioni tipiche che intercorrono tra le componenti del Design Pattern$_G$ MVC

3.2 DDDMob::Model

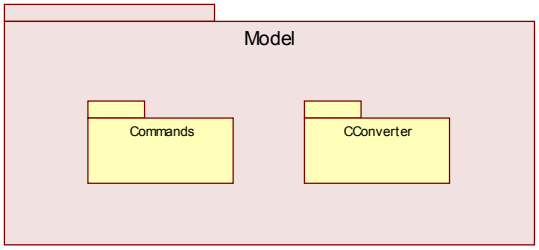

Figura 2: Componente DDDMob::Model

Package$_G$ per il componente Model dell'architettura MVC

3.3 DDDMob::Model::Commands

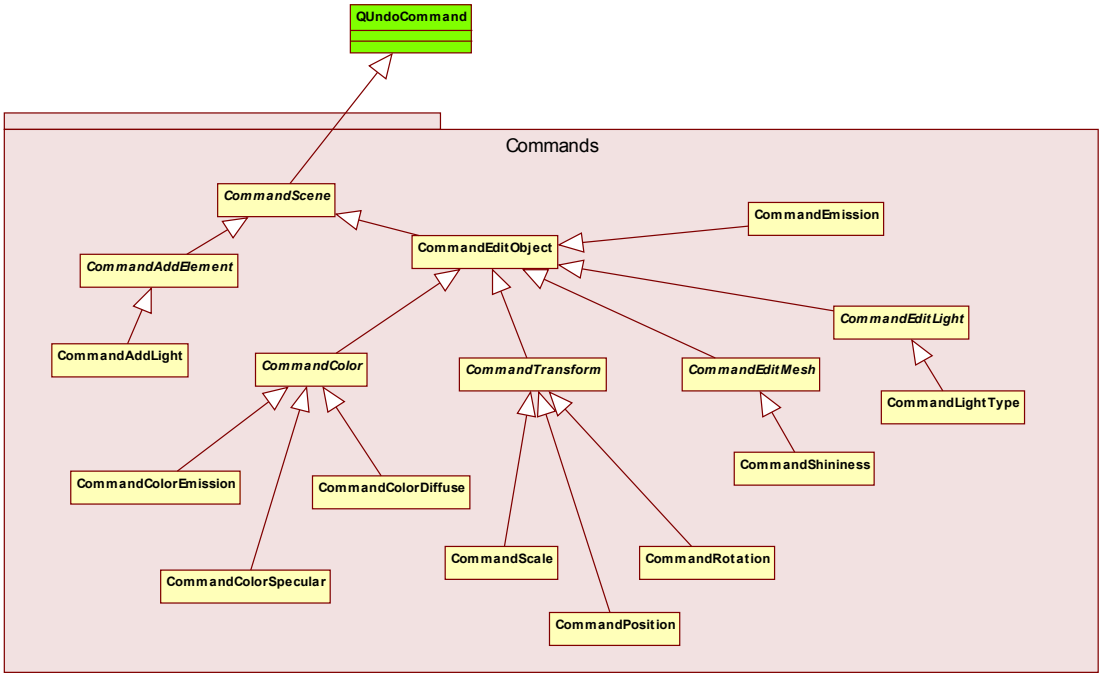

Figura 3: Componente DDDMob::Model::Commands

Componente parte del Model per la gestione dei comandi

3.3.1 CommandTransform (abstract)

CommandTransform
#transformOriginal: QVector3D #transformNew: QVector3D
+CommandTransform(scene: Scene*, sceneObject: SceneObject*, transformOriginal: QVector3D, transformNew: QVector3D)

Figura 4: Classe CommandTransform

Descrizione

Classe base per i comandi di trasformazione;

Utilizzo

Viene utilizzata per applicare un generico parametro di trasformazione ad un oggetto della scena$_G$ 3D, specificato poi nelle classi che ereditano da questa;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Ereditata da

- DDDMob :: Model :: Commands :: CommandPosition;

- DDDMob :: Model :: Commands :: CommandScale;
- DDDMob :: Model :: Commands :: CommandRotation.

Attributi

`# QVector3D transformOriginal`
Rappresenta la trasformazione originale prima di aver effettuato la nuova trasformazione;

`# QVector3D transformNew`
Rappresenta la nuova trasformazione effettuata.

Metodi

`+ CommandTransform(scene : Scene*, sceneObject : SceneObject*, transformOriginal : QVector3D, transformNew : QVector3D, )`
Costruttore di CommandTransform
Argomenti

- scene : Scene*
  La scena$_G$ che contiene l'oggetto da trasformare;
- sceneObject : SceneObject*
  L'oggetto della scena$_G$ su cui applicare la trasformazione;
- transformOriginal : QVector3D
  Il vettore che contiene i parametri iniziali della trasformazione;
- transformNew : QVector3D
  Il vettore che contiene i parametri finali della trasformazione.

3.3.2 CommandAddLight (class)

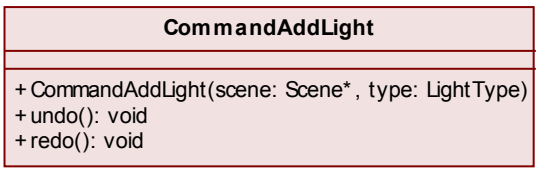

Figura 5: Classe CommandAddLight

Descrizione
Classe che rappresenta il comando di aggiunta di una luce con i parametri di default alla scena$_G$ 3D;

Utilizzo
Viene utilizzata per gestire i Signal$_G$ riguardanti la creazione di una luce ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandAddElement.

Metodi

`+ CommandAddLight(scene : Scene*, type : LightType, )`

costruttore di CommandAddLight

Argomenti

- scene : Scene*
  La scena$_G$ alla quale si desidera aggiungere la fonte di luce;
- type : LightType
  La tipologia di luce da aggiungere alla scena$_G$.

`+ void undo()`

Annulla il comando

Note

- Questo metodo è stato ridefinito.

`+ void redo()`

Ripristina il comando annullato

Note

- Questo metodo è stato ridefinito.

3.3.3 CommandEditMesh (abstract)

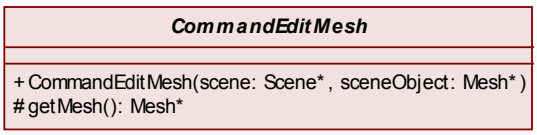

Figura 6: Classe CommandEditMesh

Descrizione

Classe che rappresenta il comando di modifica di una mesh;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica di una mesh ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Ereditata da

- DDDMob :: Model :: Commands :: CommandShininess.

Metodi

`+ CommandEditMesh(scene : Scene*, sceneObject : Mesh*, )`

costruttore per CommandEditMesh

Argomenti

- scene : Scene*
  La scena$_G$ contenente la mesh che si desidera modificare;
- sceneObject : Mesh*
  La mesh che si desidera modificare.

`# Mesh* getMesh()`
getter della Mesh

3.3.4 CommandEditLight (abstract)

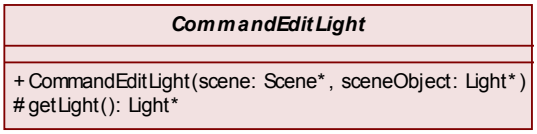

Figura 7: Classe CommandEditLight

Descrizione
Classe che rappresenta il comando di modifica della luce selezionata;

Utilizzo
Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica della luce selezionata ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Ereditata da

- DDDMob :: Model :: Commands :: CommandLightType.

Metodi

`+ CommandEditLight(scene : Scene*, sceneObject : Light*, )`
Costruttore di CommandEditLight
Argomenti

- scene : Scene*
  La scena$_G$ contenente la luce che si desidera modificare;
- sceneObject : Light*
  La luce che si desidera modificare.

`# Light* getLight()`
getter di CommandEditLight

3.3.5 CommandScene (abstract)

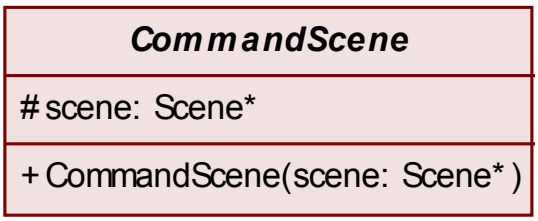

Figura 8: Classe CommandScene

Descrizione

Classe che rappresenta un generico comando da eseguire sulla scena$_G$. È il padre di tutti i possibili componenti concrete command del Design Pattern$_G$ Command;

Utilizzo

Viene utilizzata per applicare un generico comando da eseguire sulla scena$_G$ 3D, specificato poi nelle classi che ereditano da questa;

Classi ereditate

- QUndoCommand.

Ereditata da

- DDDMob :: Model :: Commands :: CommandAddElement;
- DDDMob :: Model :: Commands :: CommandEditObject.

Attributi

`# Scene* scene`

Riferimento alla scena$_G$ del modello.

Metodi

`+ CommandScene(scene : Scene*, )`

Costruttore di CommandScene

Argomenti

- scene : Scene*
  La scena$_G$ sulla quale verrà eseguito il comando.

3.3.6 CommandAddElement (abstract)

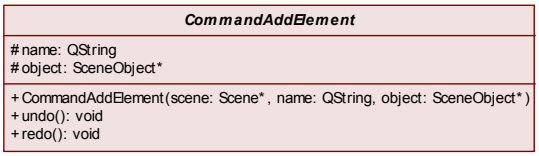

Figura 9: Classe CommandAddElement

Descrizione

Classe che rappresenta un comando di aggiunta di un elemento alla scena$_G$ 3D;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti l'aggiunta di un elemento ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandScene.

Ereditata da

- DDDMob :: Model :: Commands :: CommandAddLight.

Attributi

`# QString name`
Nome dell'elemento;

`# SceneObject* object`
Oggetto della scena$_G$.

Metodi

`+ CommandAddElement(scene : Scene*, name : QString , object : SceneObject*, )`
costruttore CommandAddElement
Argomenti

- scene : Scene*
  La scena$_G$ nella quale si desidera aggiungere l'elemento;
- name : QString
  Il nome dell'oggetto da aggiungere;
- object : SceneObject*
  L'oggetto da aggiungere alla scena$_G$.

`+ void undo()`
Annulla il comando
Note

- Questo metodo è stato ridefinito.

`+ void redo()`
Ripristina il comando annullato
Note

- Questo metodo è stato ridefinito.

3.3.7 CommandPosition (class)

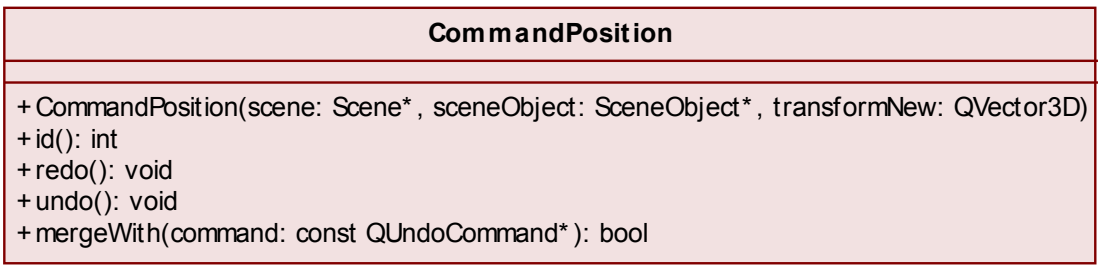

Figura 10: Classe CommandPosition

Descrizione

Classe che rappresenta un comando per cambiare la posizione dell'oggetto della scena$_G$ 3D selezionato;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica della posizione di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandTransform.

Metodi

`+ CommandPosition(scene : Scene*, sceneObject : SceneObject*, transformNew : QVector3D, )`
Costruttore di CommandPosition
Argomenti

- scene : Scene*
  La scena$_G$ contenente l'oggetto del quale si vuole modificare la posizione;
- sceneObject : SceneObject*
  L'oggetto del quale si vuole modificare la posizione;
- transformNew : QVector3D
  La nuova posizione dell'oggetto della scena$_G$.

`+ int id()`
Restituisce l'id del comando come specificato nell'appendice C
Note

- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ void redo()`
Metodo che esegue nuovamente l'ultimo comando annullato
Note

- Questo metodo è stato ridefinito.

`+ void undo()`
Metodo che annulla l'ultimo comando eseguito
Note

- Questo metodo è stato ridefinito.

`+ bool mergeWith(command : const QUndoCommand*, )`
Metodo che prova ad eseguire un merge$_G$ del comando corrente con quello passato come parametro. Se il comando passato non ha lo stesso id del comando corrente (si veda l'appendice C), l'operazione fallisce
Argomenti

- command : const QUndoCommand*
  Il comando con il quale è richiesto provare il merge$_G$.

Note

- Questo metodo è stato ridefinito.

3.3.8 CommandScale (class)

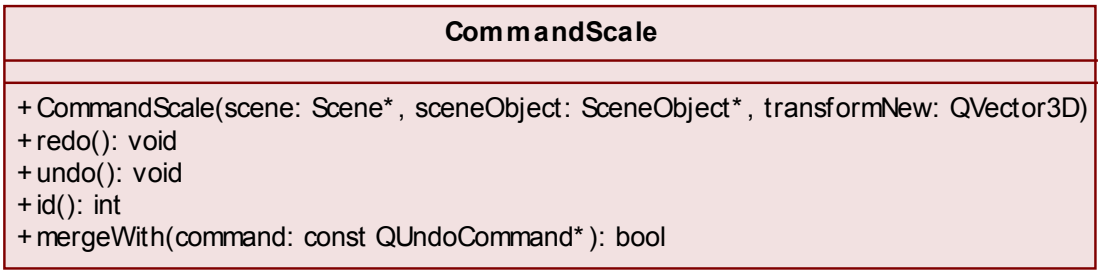

Figura 11: Classe CommandScale

Descrizione

Classe che rappresenta un comando per cambiare la dimensione dell'oggetto della scena$_G$ 3D selezionato;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la dimensione di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandTransform.

Metodi

`+ CommandScale(scene : Scene*, sceneObject : SceneObject*, transformNew : QVector3D, )`

Costruttore di CommandScale

Argomenti

- scene : Scene*
  La scena$_G$ che contiene l'oggetto che si desidera ridimensionare;
- sceneObject : SceneObject*
  L'oggetto che si desidera ridimensionare;
- transformNew : QVector3D
  Vettore contenente la trasformazione.

`+ void redo()`

Metodo che esegue nuovamente l'ultimo comando annullato

Note

- Questo metodo è stato ridefinito.

`+ void undo()`

Metodo che annulla l'ultimo comando eseguito

Note

- Questo metodo è stato ridefinito.

`+ int id()`

Restituisce l'id del comando come specificato nell'appendice C

Note

- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ bool mergeWith(command : const QUndoCommand*, )`

Metodo che prova ad eseguire un merge$_G$ del comando corrente con quello passato come parametro. Se il comando passato non ha lo stesso id del comando corrente (si veda l'appendice C), l'operazione fallisce

Argomenti

- command : const QUndoCommand*
  Il comando con il quale è richiesto provare il merge$_G$.

Note

- Questo metodo è stato ridefinito.

3.3.9 CommandRotation (class)

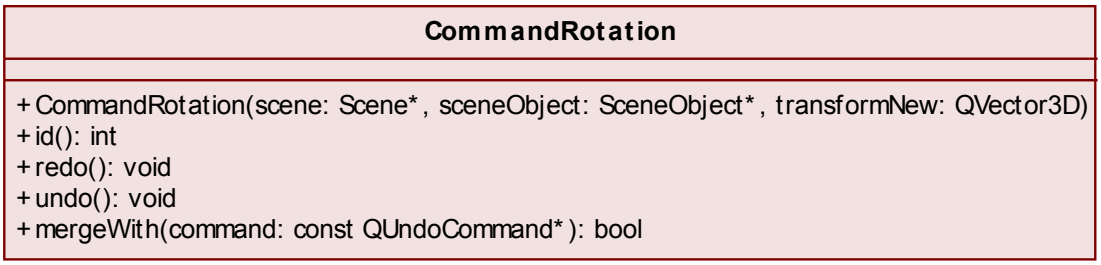

Figura 12: Classe CommandRotation

Descrizione

Classe che rappresenta un comando per cambiare la rotazione del modello;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la rotazione di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandTransform.

Metodi

`+ CommandRotation(scene : Scene*, sceneObject : SceneObject*, transformNew : QVector3D, )`

Costruttore di CommandRotation

Argomenti

- scene : Scene*
  La scena$_G$ contenente l'oggetto che si desidera ruotare;
- sceneObject : SceneObject*
  L'oggetto che si desidera ruotare;
- transformNew : QVector3D
  Vettore contenente la trasformazione da applicare.

`+ int id()`
Restituisce l'id del comando come specificato nell'appendice C
Note
- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ void redo()`
Metodo che esegue nuovamente l'ultimo comando annullato
Note
- Questo metodo è stato ridefinito.

`+ void undo()`
Metodo che annulla l'ultimo comando eseguito
Note
- Questo metodo è stato ridefinito.

`+ bool mergeWith(command : const QUndoCommand*, )`
Metodo che prova ad eseguire un merge$_G$ del comando corrente con quello passato come parametro. Se il comando passato non ha lo stesso id del comando corrente (si veda l'appendice C), l'operazione fallisce
Argomenti
- command : const QUndoCommand*
  Il comando con il quale è richiesto provare il merge$_G$.

Note
- Questo metodo è stato ridefinito.

3.3.10 CommandLightType (class)

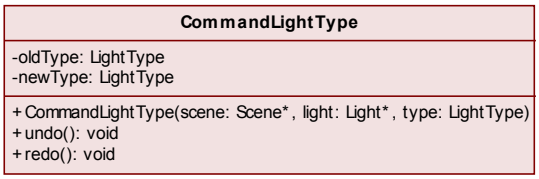

Figura 13: Classe CommandLightType

Descrizione
Classe che rappresenta un comando per cambiare il tipo della luce selezionata;

Utilizzo
Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica della tipologia di una luce ed invocare i corretti metodi del Model;

Classi ereditate
- DDDMob :: Model :: Commands :: CommandEditLight.

Attributi

`- LightType oldType`
Vecchio tipo di luce;

`- LightType newType`
Nuovo tipo di luce.

Metodi

`+ CommandLightType(scene : Scene*, light : Light*, type : LightType, )`
Costruttore di CommandLightType
Argomenti

- scene : Scene*
  La scena$_G$ contenente la luce da modificare;
- light : Light*
  La luce di cui si desidera modificare il tipo;
- type : LightType
  Il tipo di luce da impostare.

`+ void undo()`
Elimina il comando
Note

- Questo metodo è stato ridefinito.

`+ void redo()`
Ripristina il comando annullato
Note

- Questo metodo è stato ridefinito.

3.3.11 CommandShininess (class)

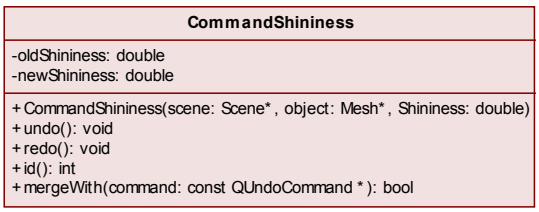

Figura 14: Classe CommandShininess

Descrizione
Classe che rappresenta il comando che cambia il valore di lucentezza della mesh;

Utilizzo
Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica della lucentezza di una mesh ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditMesh.

Attributi

`- double oldShininess`
Vecchio valore di lucentezza;

`- double newShininess`
Nuovo valore di lucentezza.

Metodi

`+ CommandShininess(scene : Scene* , object : Mesh* , Shininess : double, )`
costruttore CommandShininess
Argomenti

- scene : Scene*
  La scena$_G$ che contiene l'oggetto del quale si vuole modificare la lucentezza;
- object : Mesh*
  L'oggetto del quale si vuole modificare la lucentezza;
- Shininess : double
  Il valore di lucentezza che si desidera venga applicato.

`+ void undo()`
Annulla il comando
Note

- Questo metodo è stato ridefinito.

`+ void redo()`
ripristina il comando annullato
Note

- Questo metodo è stato ridefinito.

`+ int id()`
Restituisce l'id del comando come specificato nell'appendice C
Note

- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ bool mergeWith(command : const QUndoCommand * , )`
Metodo che prova ad eseguire un merge$_G$ del comando corrente con quello passato come parametro. Se il comando passato non ha lo stesso id del comando corrente (si veda l'appendice C), l'operazione fallisce
Argomenti

- command : const QUndoCommand *
  Il comando con il quale è richiesto provare il merge$_G$.

Note

- Questo metodo è stato ridefinito.

3.3.12 CommandEmission (class)

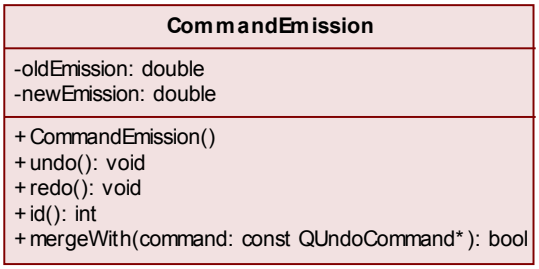

Figura 15: Classe CommandEmission

Descrizione

Classe che rappresenta il comando per cambiare il valore della quantità di emissione del materiale di un oggetto;

Utilizzo

Viene utilizzata per gestire i Signal_G riguardanti la quantità di emissione del materiale di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Attributi

`- double oldEmission`
Valore del colore di emissione prima della modifica;

`- double newEmission`
Valore del colore di emissione dopo la modifica.

Metodi

`+ CommandEmission()`
Costruttore di CommandEmission

`+ void undo()`
Metodo che annulla l'ultimo comando eseguito
Note
- Questo metodo è stato ridefinito.

`+ void redo()`
Metodo che esegue nuovamente l'ultimo comando annullato
Note
- Questo metodo è stato ridefinito.

`+ int id()`
Restituisce l'id del comando come specificato nell'appendice C
Note
- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ bool mergeWith(command : const QUndoCommand*, )`

Metodo che prova ad eseguire un merge$_G$ del comando corrente con quello passato come parametro. Se il comando passato non ha lo stesso id del comando corrente (si veda l'appendice C), l'operazione fallisce

Argomenti

- command : const QUndoCommand*
  Il comando con il quale è richiesto provare il merge$_G$.

Note

- Questo metodo è stato ridefinito.

3.3.13 CommandColor (abstract)

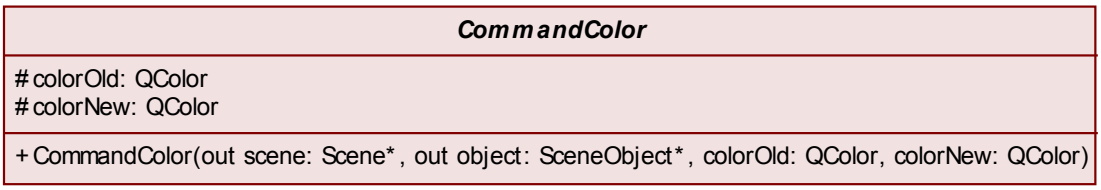

Figura 16: Classe CommandColor

Descrizione

Classe che rappresenta il comando per cambiare le caratteristiche legate al colore dell'oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti il colore dell'oggetto;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Ereditata da

- DDDMob :: Model :: Commands :: CommandColorEmission;
- DDDMob :: Model :: Commands :: CommandColorSpecular;
- DDDMob :: Model :: Commands :: CommandColorDiffuse.

Attributi

`# QColor colorOld`

Colore vecchio dell'oggetto;

`# QColor colorNew`

Rappresenta il colore nuovo dell'oggetto.

Metodi

`+ CommandColor(scene : Scene*, object : SceneObject*, colorOld : QColor, colorNew : QColor, )`

Costruttore della classe CommandColor

Argomenti

- scene : Scene*
  La scena$_G$ contenente l'oggetto da modificare;
- object : SceneObject*
  L'oggetto della scena$_G$ da modificare;
- colorOld : QColor
  Il colore precedentemente assegnato all'oggetto;
- colorNew : QColor
  Il colore che si desidera assegnare all'oggetto.

3.3.14 CommandColorEmission (class)

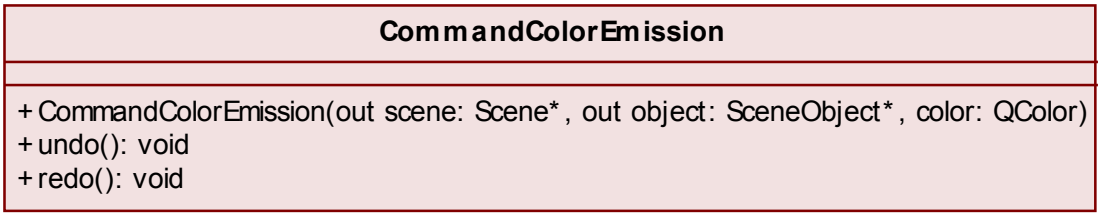

Figura 17: Classe CommandColorEmission

Descrizione
Classe che rappresenta il comando per cambiare il colore di emissione dell'oggetto;

Utilizzo
Viene utilizzata per gestire i Signal$_G$ riguardanti il colore di emissione dell'oggetto;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandColor.

Metodi

`+ CommandColorEmission(scene : Scene*, object : SceneObject*, color : QColor, )`
Costruttore della classe CommandColorEmission
Argomenti
- scene : Scene*
  La scena$_G$ che contiene l'oggetto di cui si desidera modificare l'emissività;
- object : SceneObject*
  L'oggetto di cui si desidera modificare l'emissività;
- color : QColor
  Il colore di emissività da assegnare all'oggetto.

`+ void undo()`
Ripristina la modifica precedente
Note
- Questo metodo è stato ridefinito.

`+ void redo()`
Ripristina l'ultima modifica annullata
Note

- Questo metodo è stato ridefinito.

3.3.15 CommandColorSpecular (class)

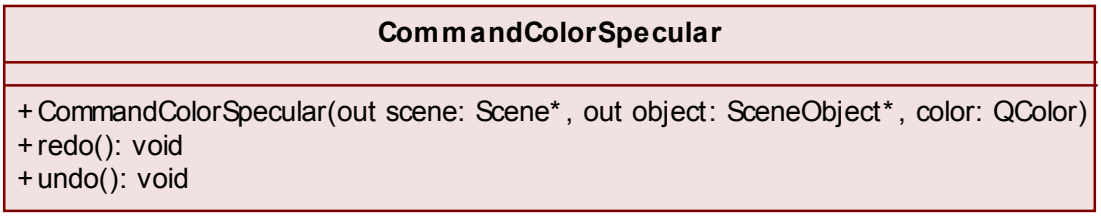

Figura 18: Classe CommandColorSpecular

Descrizione

Classe che rappresenta il comando per cambiare il colore di diffusione$_G$ dell'oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti il colore di diffusione$_G$ dell'oggetto;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandColor.

Metodi

`+ CommandColorSpecular(scene : Scene*, object : SceneObject*, color : QColor, )`

Costruttore della classe CommandColorSpecular

Argomenti

- scene : Scene*
  La scena$_G$ contenente l'oggetto di cui si vuole modificare il colore speculare$_G$;
- object : SceneObject*
  L'oggetto della scena$_G$ di cui si desidera modificare il colore speculare$_G$;
- color : QColor
  Il colore speculare$_G$ che si desidera assegnare all'oggetto.

`+ void redo()`

Ripristina l'ultima modifica annullata

Note

- Questo metodo è stato ridefinito.

`+ void undo()`

Ripristina la modifica precedente

Note

- Questo metodo è stato ridefinito.

3.3.16 CommandColorDiffuse (class)

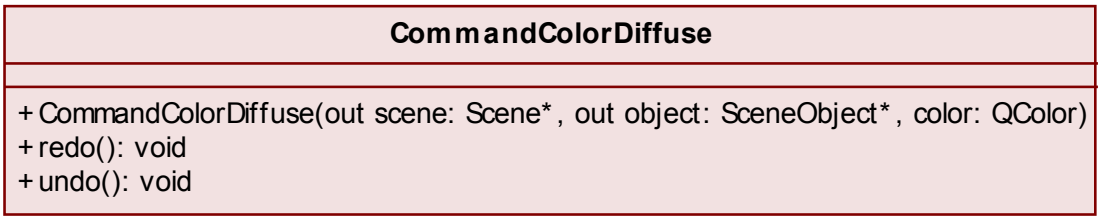

Figura 19: Classe CommandColorDiffuse

Descrizione

Comando per cambiare il colore di diffusione$_G$ dell'oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti il colore di diffusione$_G$ dell'oggetto;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandColor.

Metodi

`+ CommandColorDiffuse(scene : Scene*, object : SceneObject*, color : QColor, )`

Costruttore della classe CommandColorDiffuse

Argomenti

- scene : Scene*
  La scena$_G$ contenente l'oggetto di cui si vuole modificare il colore di diffusione$_G$;
- object : SceneObject*
  L'oggetto di cui si vuole modificare il colore di diffusione$_G$;
- color : QColor
  Il colore di diffusione$_G$ che si desidera assegnare all'oggetto.

`+ void redo()`

Ripristina l'ultima modifica annullata

Note

- Questo metodo è stato ridefinito.

`+ void undo()`

Ripristina la modifica precedente

Note

- Questo metodo è stato ridefinito.

3.3.17 CommandEditObject (class)

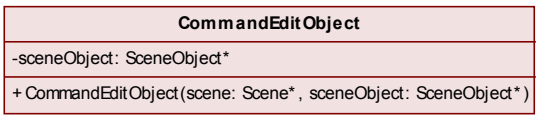

Figura 20: Classe CommandEditObject

Descrizione

Classe che rappresenta il comando di modifica di un oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandScene.

Ereditata da

- DDDMob :: Model :: Commands :: CommandTransform;
- DDDMob :: Model :: Commands :: CommandEditMesh;
- DDDMob :: Model :: Commands :: CommandEditLight;
- DDDMob :: Model :: Commands :: CommandEmission;
- DDDMob :: Model :: Commands :: CommandColor.

Attributi

```
- SceneObject* sceneObject
```

Riferimento all'oggetto della scena$_G$.

Metodi

```
+ CommandEditObject(scene : Scene*, sceneObject : SceneObject*, )
```

Costruttore di CommandEditObject

Argomenti

- scene : Scene*
  La scena$_G$ contenente l'oggetto da modificare;
- sceneObject : SceneObject*
  L'oggetto da modificare.

3.4 DDDMob::Model::CConverter

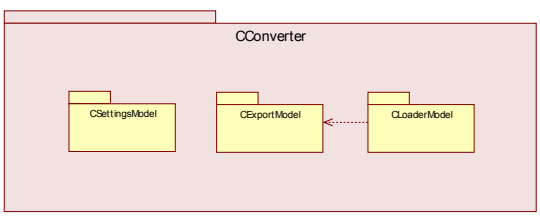

Figura 21: Componente DDDMob::Model::CConverter

Macro componente per l'integrazione delle funzionalità di importazione, applicazione dei limiti impostati ed esportazione a formare il convertitore di base

3.5 DDDMob::Model::CConverter::CSettingsModel

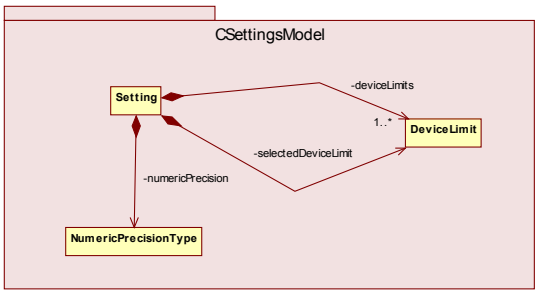

Figura 22: Componente DDDMob::Model::CConverter::CSettingsModel

Componente parte del Model per le funzionalità di impostazioni e configurazione

3.5.1 Setting (class)

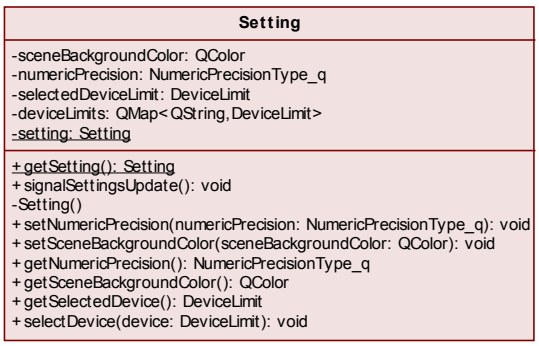

Figura 23: Classe Setting

Descrizione

Classe implementata con il Design Pattern$_G$ Singleton che contiene tutte le impostazioni del programma;

Utilizzo

Viene utilizzata per modificare e recuperare i limiti di importazione, precisione di esportazione e colore di sfondo. Riceve richieste di modifica di alcuni elementi o di restituzione dei dati, e notifica alla vista quando gli stessi sono disponibili o aggiornati;

Attributi

`- QColor sceneBackgroundColor`
: Colore di sfondo della scena$_G$;

`- NumericPrecisionType numericPrecision`
: Precisione numerica dell'oggetto esportato;

`- DeviceLimit selectedDeviceLimit`
Limiti del dispositivo selezionato;

`- QMap<QString,DeviceLimit> deviceLimits`
Limiti dei dispositivi predefiniti;

`- Setting setting`
Riferimento a se stesso per l'accesso protetto.

Metodi

`+ Setting& getSetting()`
Ritorna l'istanza dell'oggetto Setting in modo protetto, secondo il Design Pattern$_G$ Singleton
Note
- Deve essere un metodo statico.

`# void signalSettingsUpdate()`
Notifica l'avvenuto aggiornamento delle impostazioni

`- Setting()`
Costruttore privato dell'oggetto Setting

`+ void setNumericPrecision(numericPrecision : NumericPrecisionType, )`

Imposta la precisione numerica dei dati che si desidera per l'esportazione della scena$_G$
Argomenti
- numericPrecision : NumericPrecisionType
  Precisione numerica da impostare.

`+ void setSceneBackgroundColor(sceneBackgroundColor : QColor, )`
Imposta il colore di sfondo dell'anteprima della scena$_G$
Argomenti
- sceneBackgroundColor : QColor
  Colore della scena$_G$.

`+ NumericPrecisionType getNumericPrecision()`
Ritorna la precisione numerica correntemente impostata

`+ QColor getSceneBackgroundColor()`
Ritorna il colore di sfondo dell'anteprima della scena$_G$ correntemente impostato

`+ DeviceLimit& getSelectedDevice()`
Ritorna un riferimento al dispositivo selezionato

`+ void selectDevice(device : QString, )`
Seleziona un device come device correntemente selezionato
Argomenti
- device : QString
  Dispositivo da impostare.

`- void loadSettings()`
Carica le impostazioni dal file settings.ini

`+ void saveSettings()`
Salva le impostazioni nel file settings.ini

`# void signalError()`
Notifica l'errore nell'importazione dei valori

`# void signalBackgroundColorUpdate()`
Notifica l'avvenuto aggiornamento dell'impostazione di colore di sfondo

`# void signalNumericPrecisionUpdate()`
Notifica l'avvenuto aggiornamento dell'impostazione di precisione numerica

`+ const QMap<QString,DeviceLimit>& getDeviceLimits()`
Ritorna i limiti dei dispositivi
Note
- Deve essere esplicitamente marcato come costante.

3.5.2 DeviceLimit (class)

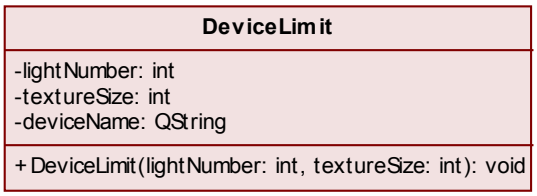

Figura 24: Classe DeviceLimit

Descrizione
Classe che rappresenta i limiti di un dispositivo. Durante il caricamento del programma viene caricata da un file di configurazione la lista dei dispositivi supportati dal programma e i relativi limiti;

Utilizzo
Viene utilizzata per salvare i dati che rappresentano i limiti di esportazione relativi a un dispositivo;

Attributi

`- int lightNumber`
Massimo numero di luci ammesse nella scena$_G$;

`- int textureSize`
Massima dimensione delle texture ammissibile;

`- QString deviceName`
Il nome del dispositivo del quale si vogliono rappresentare i limiti di OpenGL ES$_G$ 2.0. Viene letto da un file di configurazione all'avvio del programma.

Metodi

`+ void DeviceLimit(lightNumber : int, textureSize : int, )`

Costruttore della classe DeviceLimit

Argomenti

- lightNumber : int
  Massimo numero di luci ammesse;
- textureSize : int
  Massima dimensione delle texture ammissibile.

3.5.3 NumericPrecisionType (class)

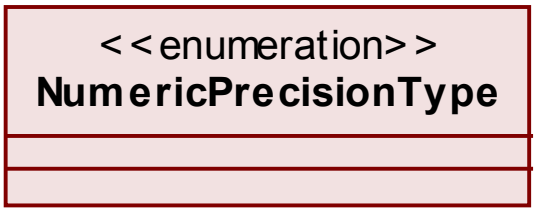

Figura 25: Classe NumericPrecisionType

Descrizione

Classe che contiene le diverse tipologie di precisione dei numeri. Le precisioni disponibili sono float e double;

Utilizzo

Viene utilizzata come `Enumerate`.
I possibili valori sono:

- FLOAT;
- DOUBLE.

3.6 DDDMob::Model::CConverter::CExportModel

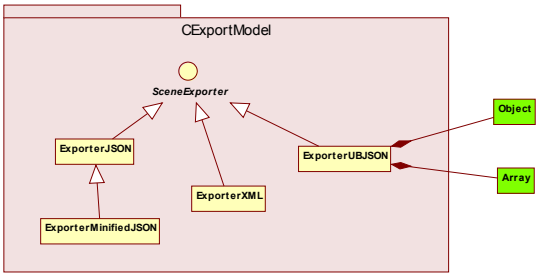

Figura 26: Componente DDDMob::Model::CConverter::CExportModel

Componente parte del Model per le funzionalità di esportazione

3.6.1 SceneExporter (interface)

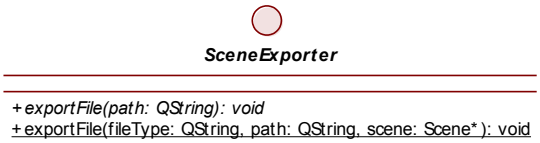

Figura 27: Classe SceneExporter

Descrizione

Interfaccia per la componente strategy del Design Pattern$_G$ Strategy per la selezione dell'algoritmo di esportazione della scena$_G$ tra tutti i disponibili;

Utilizzo

Permette di selezionare dinamicamente ed in modo estensibile l'algoritmo di esportazione della scena$_G$ e di conseguenza il formato del file salvato in uscita;

Ereditata da

- DDDMob :: Model :: CConverter :: CExportModel :: ExporterUBJSON;
- DDDMob :: Model :: CConverter :: CExportModel :: ExporterXML;
- DDDMob :: Model :: CConverter :: CExportModel :: ExporterJSON.

Metodi

`+ void exportFile(path : QString, )`

Esporta un file dato il percorso. Metodo virtuale ed astratto, deve essere ridefinito dalle classi che ereditano da questa, implementando la funzionalità di salvataggio

Argomenti

- path : QString
  Percorso dove esportare il file convertito.

Note

- Deve essere astratto;
- Questo metodo è stato ridefinito.

`+ void exportFile(fileType : QString, path : QString, scene : Scene*, )`

Metodo statico che, dato il percorso del file da esportare e l'estensione del formato desiderato, istanzia la classe corretta per esportare la scena$_G$ e ne chiama il metodo exportFile per realizzare la funzione richiesta

Argomenti

- fileType : QString
  Il formato nel quale si desidera esportare la scena$_G$;
- path : QString
  Il percorso dove si desidera salvare il file;
- scene : Scene*
  Riferimento alla scena$_G$ contenente gli oggetti da esportare.

Note

- Deve essere un metodo statico.

3.6.2 ExporterUBJSON (class)

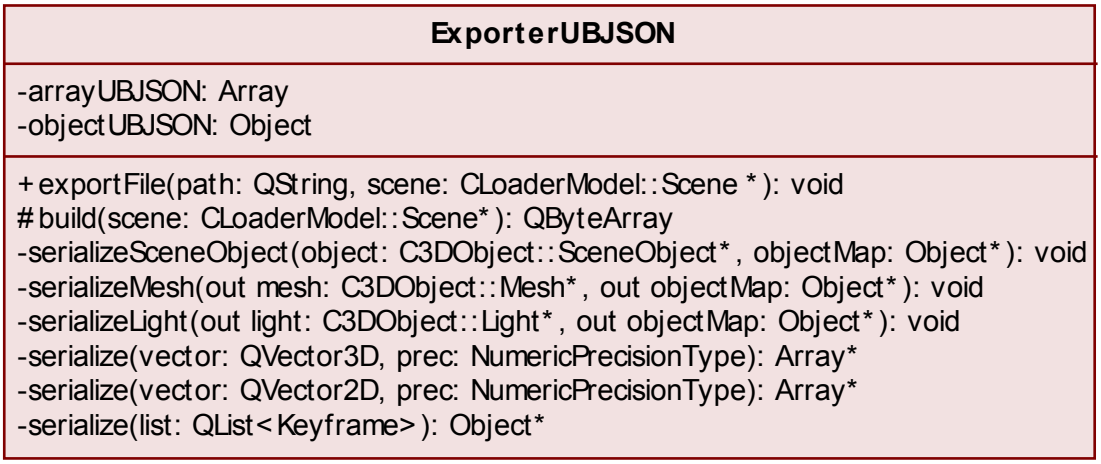

Figura 28: Classe ExporterUBJSON

Descrizione

Classe che rappresenta un algoritmo di esportazione della scena$_G$ in un file di formato UBJSON$_G$. È uno dei componenti concrete component del Design Pattern$_G$ Strategy;

Utilizzo

Viene utilizzata per creare un file UBJSON$_G$ a partire dalla scena$_G$ 3D. Per creare il file UBSJON viene utilizzata la libreria ubjson-cpp;

Classi ereditate

- DDDMob :: Model :: CConverter :: CExportModel :: SceneExporter.

Attributi

`- Array arrayUBJSON`

Array contenente gli elementi della scena$_G$ da serializzare;

`- Object objectUBJSON`

Oggetto contenente le animazioni della scena$_G$ da serializzare.

Metodi

`+ void exportFile(path : QString, scene : CLoaderModel::Scene *, )`

Esporta un file dato il percorso

Argomenti

- path : QString
  Percorso dove esportare il file convertito;
- scene : CLoaderModel::Scene *
  La scena$_G$ che si desidera esportare.

Note

- Questo metodo è stato ridefinito.

`# QByteArray build(scene : CLoaderModel::Scene*, )`

Crea lo stream$_G$ di dati per il file UBJSON$_G$

Argomenti

- scene : CLoaderModel::Scene*
  Riferimento alla scena$_G$ da serializzare.

Note

- Deve essere esplicitamente marcato come costante.

`- void serializeSceneObject(object : C3DObject::SceneObject*, objectMap : Object*, )`

Funzione di utilità, aggiunge a ObjectMap gli attributi propri di SceneObject

Argomenti

- object : C3DObject::SceneObject*
  L'oggetto da serializzare;
- objectMap : Object*
  La mappa di oggetti su cui inserire le informazioni.

Note

- Deve essere esplicitamente marcato come costante.

`- void serializeMesh(mesh : C3DObject::Mesh*, objectMap : Object*, )`

Funzione d'utilità, aggiunge a ObjectMap gli attributi propri dell'oggetto Mesh

Argomenti

- mesh : C3DObject::Mesh*
  La mesh da serializzare;
- objectMap : Object*
  La mappa di oggetti su cui inserire le informazioni.

Note

- Deve essere esplicitamente marcato come costante.

`- void serializeLight(light : C3DObject::Light*, objectMap : Object*, )`

Funzione d'utilità, aggiunge a ObjectMap gli attributi propri dell'oggetto Light

Argomenti

- light : C3DObject::Light*
  La luce da serializzare;
- objectMap : Object*
  La mappa di oggetti su cui inserire le informazioni.

Note

- Deve essere esplicitamente marcato come costante.

`- Array* serialize(vector : QVector3D, prec : NumericPrecisionType, )`

Funzione d'utilità, serializza un QVector3D e ritorna un Array della libreria UBJSON$_G$

Argomenti

- vector : QVector3D
  Il vettore da serializzare;

- prec : NumericPrecisionType
  La precisione numerica da utilizzare nell'esportazione.

Note

- Deve essere esplicitamente marcato come costante.

`- Array* serialize(vector : QVector2D, prec : NumericPrecisionType, )`

Funzione d'utilità, serializza un QVector2D e ritorna un Array della libreria UBJSON$_G$

Argomenti

- vector : QVector2D
  Il vettore da serializzare;
- prec : NumericPrecisionType
  La precisione numerica da utilizzare nell'esportazione.

Note

- Deve essere esplicitamente marcato come costante.

`- Object* serialize(list : QList<Keyframe>, )`

Funzione d'utilità, serializza una QList e ritorna un Object della libreria UBJSON$_G$

Argomenti

- list : QList<Keyframe$_G$ >
  La lista di keyframe$_G$ da serializzare.

Note

- Deve essere esplicitamente marcato come costante.

3.6.3 ExporterXML (class)

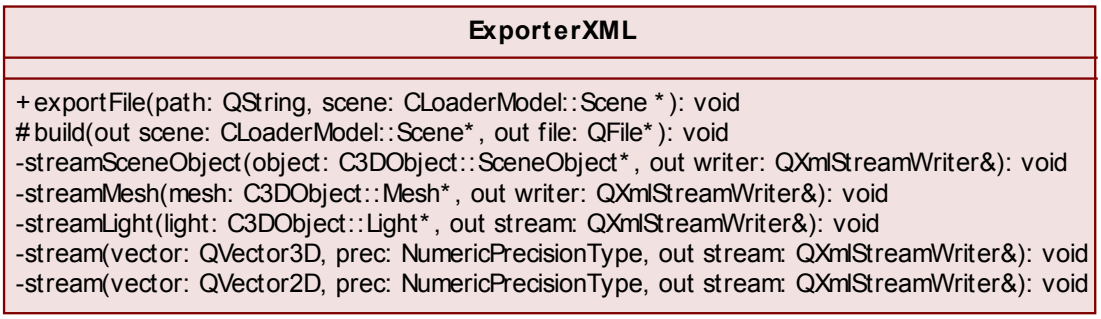

Figura 29: Classe ExporterXML

Descrizione

Classe che rappresenta un algoritmo di esportazione della scena$_G$ in un file di formato XML$_G$. È uno dei componenti concrete component del Design Pattern$_G$ Strategy;

Utilizzo

Viene utilizzata per creare un file XML$_G$ a partire dalla scena$_G$ 3D. Per la creazione del file JSON$_G$ viene utilizzata la classe QJsonDocument presente in QtXml;

Classi ereditate

- DDDMob :: Model :: CConverter :: CExportModel :: SceneExporter.

Metodi

`+ void exportFile(path : QString, scene : CLoaderModel::Scene *, )`

Scrive lo stream$_G$ di dati restituito dal metodo build in un file nel path specificato

Argomenti

- path : QString
  Percorso dove esportare il file convertito;
- scene : CLoaderModel::Scene *
  La scena$_G$ che si desidera esportare.

Note

- Questo metodo è stato ridefinito.

`# void build(scene : CLoaderModel::Scene*, file : QFile*, )`

Crea lo stream$_G$ di dati per il file in formato XML$_G$, e lo popola con i dati presenti nella scena$_G$ secondo lo XMLSchema. Per serializzare gli oggetti della scena$_G$ si appoggia ai metodi privati di utilità definiti

Argomenti

- scene : CLoaderModel::Scene*
  La scena$_G$ da esportare;
- file : QFile*
  Riferimento al file dove si vuole venga salvato l'XML$_G$.

Note

- Deve essere esplicitamente marcato come costante.

`- void streamSceneObject(object : C3DObject::SceneObject*, writer : QXmlStreamWriter&, )`

Dato uno sceneObject aggiunge i dati dell'oggetto allo stream$_G$ da esportare nell'XML$_G$

Argomenti

- object : C3DObject::SceneObject*
  L'oggetto da esportare;
- writer : QXmlStreamWriter&
  Lo stream$_G$ XML$_G$ al quale aggiungere le informazioni.

Note

- Deve essere esplicitamente marcato come costante.

`- void streamMesh(mesh : C3DObject::Mesh*, writer : QXmlStreamWriter&, )`

Data una mesh aggiunge i dati dell'oggetto allo stream$_G$ da esportare nell'XML$_G$

Argomenti

- mesh : C3DObject::Mesh*
  La mesh che si desidera inserire nello stream$_G$ XML$_G$;
- writer : QXmlStreamWriter&
  Lo stream$_G$ XML$_G$ al quale aggiungere le informazioni.

Note

- Deve essere esplicitamente marcato come costante.

`- void streamLight(light : C3DObject::Light*, stream : QXmlStreamWriter&, )`

Data una luce aggiunge i dati dell'oggetto allo stream_G da esportare nell'XML_G

Argomenti

- light : C3DObject::Light*
  La luce che si desidera inserire nello stream_G XML_G;
- stream_G : QXmlStreamWriter&
  Lo stream_G XML_G al quale aggiungere le informazioni.

Note

- Deve essere esplicitamente marcato come costante.

`- void stream(vector : QVector3D, prec : NumericPrecisionType, stream : QXmlStreamWriter&, )`

Metodo privato di utilità. Scrive su uno stream_G i campi dati di un vettore 3D nell'XML_G da esportare

Argomenti

- vector : QVector3D
  Il vettore che si desidera inserire nell'XML_G;
- prec : NumericPrecisionType
  La precisione numerica che si desidera applicare ai dati;
- stream_G : QXmlStreamWriter&
  Lo stream_G XML_G al quale aggiungere le informazioni.

Note

- Deve essere esplicitamente marcato come costante.

`- void stream(vector : QVector2D, prec : NumericPrecisionType, stream : QXmlStreamWriter&, )`

Metodo privato di utilità. Scrive su uno stream_G i campi dati di un vettore 2D nell'XML_G da esportare

Argomenti

- vector : QVector2D
  Il vettore che si desidera inserire nell'XML_G;
- prec : NumericPrecisionType
  La precisione numerica che si desidera applicare ai dati;
- stream_G : QXmlStreamWriter&
  Lo stream_G XML_G al quale aggiungere le informazioni.

Note

- Deve essere esplicitamente marcato come costante.

3.6.4 ExporterMinifiedJSON (class)

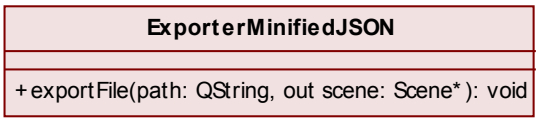

Figura 30: Classe ExporterMinifiedJSON

Descrizione

Classe che rappresenta un algoritmo di esportazione della scena$_G$ in un file di formato JSON$_G$ che contiene JSON$_G$ minificato$_G$. È uno dei componenti concrete component del Design Pattern$_G$ Strategy;

Utilizzo

Viene utilizzata per creare un file JSON$_G$ che contiene JSON$_G$ minificato$_G$ a partire dalla scena$_G$ 3D. Per la creazione del file JSON$_G$ da minificare viene utilizzata la classe ExporterJSON;

Classi ereditate

- DDDMob :: Model :: CConverter :: CExportModel :: ExporterJSON.

Metodi

`+ void exportFile(path : QString, scene : Scene*, )`

Scrive lo stream$_G$ di dati restituito dal metodo build in un file nel path specificato. È richiesto l'utilizzo del metodo build di ExporterJSON per ottenere il JSON$_G$ da minificare

Argomenti

- path : QString
  Percorso dove esportare il file convertito;
- scene : Scene*
  La scena$_G$ che si desidera esportare.

Note

- Questo metodo è stato ridefinito.

3.6.5 ExporterJSON (class)

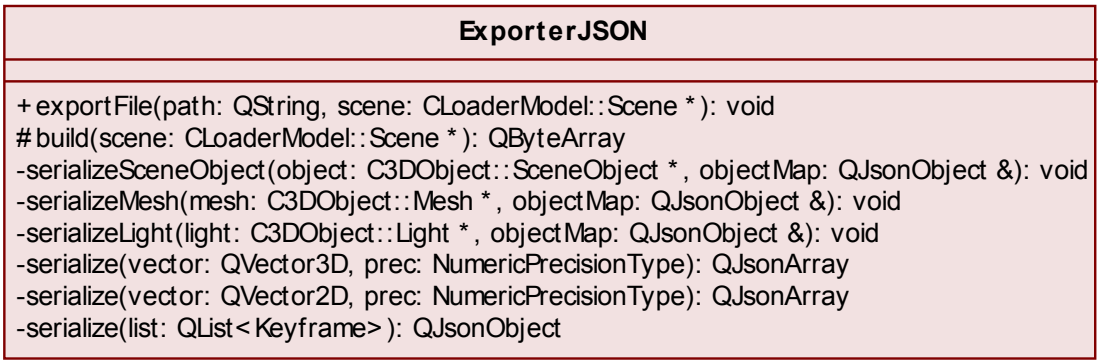

Figura 31: Classe ExporterJSON

Descrizione

Classe che rappresenta un algoritmo di esportazione della scena$_G$ in un file di formato JSON$_G$. È uno dei componenti concrete component del Design Pattern$_G$ Strategy;

Utilizzo

Viene utilizzata per creare un file JSON$_G$ a partire dalla scena$_G$ 3D. Per la creazione del file JSON$_G$ viene utilizzata la classe QJsonDocument presente in QtCore;

Classi ereditate

- DDDMob :: Model :: CConverter :: CExportModel :: SceneExporter.

Ereditata da

- DDDMob :: Model :: CConverter :: CExportModel :: ExporterMinifiedJSON.

Metodi

`+ void exportFile(path : QString , scene : CLoaderModel::Scene * , )`

Scrive lo stream$_G$ di dati restituito dal metodo build in un file nel path specificato

Argomenti

- path : QString
  Percorso dove esportare il file convertito;
- scene : CLoaderModel::Scene *
  La scena$_G$ che si desidera esportare.

Note

- Questo metodo è stato ridefinito.

`# QByteArray build(scene : CLoaderModel::Scene * , )`

Crea lo stream$_G$ di dati per il file in formato JSON$_G$, e lo popola con i dati presenti nella scena$_G$ secondo il JSONSchema. Per serializzare gli oggetti della scena$_G$ si appoggia ai metodi privati di utilità definiti

Argomenti

- scene : CLoaderModel::Scene *
  Riferimento alla scena$_G$ contenente gli oggetti da serializzare.

Note

- Deve essere esplicitamente marcato come costante.

`- void serializeSceneObject(object : C3DObject::SceneObject * , objectMap : QJsonObject &, )`

Metodo di utilità. Serializza i dati del SceneObject dato

Argomenti

- object : C3DObject::SceneObject *
  L'oggetto della scena$_G$ da serializzare;
- objectMap : QJsonObject &
  La mappa di oggetti dove si vuole siano inserite le informazioni sull'oggetto.

Note

- Deve essere esplicitamente marcato come costante.

- `void serializeMesh(mesh : C3DObject::Mesh *, objectMap : QJsonObject &, )`
  Metodo privato di utilità. Serializza le informazioni proprie di una Mesh in $JSON_G$ e le aggiunge all'oggetto QJsonObject dato
  Argomenti
  - mesh : C3DObject::Mesh *
    La mesh che si desidera serializzare;
  - objectMap : QJsonObject &
    La mappa di oggetti dove si desidera vengano inserite le informazioni sulla mesh.

  Note
  - Deve essere esplicitamente marcato come costante.

- `void serializeLight(light : C3DObject::Light * , objectMap : QJsonObject &, )`
  Metodo privato di utilità. Serializza i valori propri di un oggetto Light in $JSON_G$ e ne aggiunge i valori al QJsonObject dato
  Argomenti
  - light : C3DObject::Light *
    La luce che si desidera serializzare;
  - objectMap : QJsonObject &
    La mappa di oggetti dove si desidera vengano inserite le informazioni sulla luce.

  Note
  - Deve essere esplicitamente marcato come costante.

- `QJsonArray serialize(vector : QVector3D , prec : NumericPrecisionType, )`
  Metodo privato di utilità. Serializza i campi dati di un vettore 3D in $JSON_G$
  Argomenti
  - vector : QVector3D
    Il vettore che si desidera serializzare;
  - prec : NumericPrecisionType
    La precisione numerica che si desidera sia applicata ai dati.

  Note
  - Deve essere esplicitamente marcato come costante.

- `QJsonArray serialize(vector : QVector2D , prec : NumericPrecisionType , )`
  Metodo privato di utilità. Serializza i campi dati di un vettore 2D in $JSON_G$
  Argomenti
  - vector : QVector2D
    Il vettore che si desidera serializzare;
  - prec : NumericPrecisionType
    La precisione numerica che si desidera sia applicata ai dati.

  Note
  - Deve essere esplicitamente marcato come costante.

`- QJsonObject serialize(list : QList<Keyframe> , )`

Metodo privato di utilità. Serializza le proprietà di un Keyframe$_G$ in JSON$_G$

Argomenti

- list : QList<Keyframe$_G$ >
  La lista di keyframe$_G$ che si desidera serializzare.

Note

- Deve essere esplicitamente marcato come costante.

3.7 DDDMob::Model::CConverter::CLoaderModel

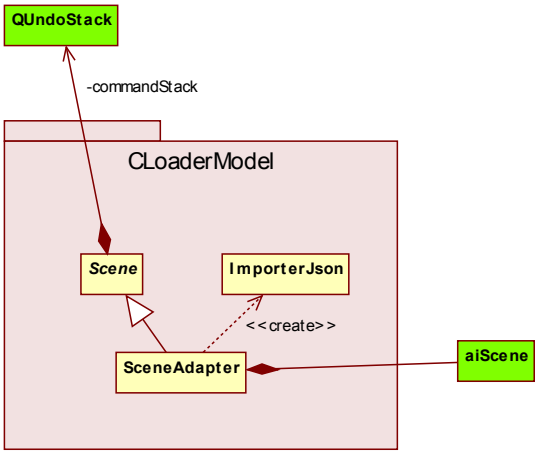

Figura 32: Componente DDDMob::Model::CConverter::CLoaderModel

Componente parte del Model per le funzionalità di caricamento del file nella scena$_G$

3.7.1 Scene (abstract)

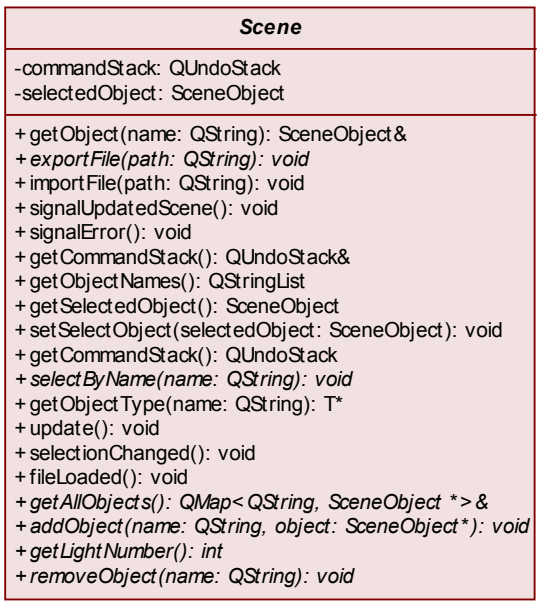

Figura 33: Classe Scene

Descrizione

Classe che rappresenta la scena$_G$ 3D che contiene le mesh e le luci. È il componente receiver del Design Pattern$_G$ Command, componente context del Design Pattern$_G$ Strategy e componente target del Design Pattern$_G$ Adapter. Internamente contiene la lista di comandi eseguiti sulla scena$_G$ 3D;

Utilizzo

Interfaccia per gestire le proprietà degli oggetti 3D. Permette la selezione di un oggetto della scena$_G$ e la sua modifica. Contiene lo stato dei comandi eseguiti sulla scena$_G$ così da poter annullare e ripristinare le modifiche effettuate. Permette l'esportazione in vari formati, selezionati grazie al Design Pattern$_G$ Strategy, e l'importazione nei vari formati permessi dalla libreria esterna utilizzata;

Ereditata da

- DDDMob :: Model :: CConverter :: CLoaderModel :: SceneAdapter.

Attributi

`- QUndoStack commandStack`
Lista delle azioni effettuate;

`- SceneObject selectedObject`
L'oggetto 3D attualmente selezionato.

Metodi

`+ SceneObject& getObject(name : QString, )`
Ritorna il riferimento all'oggetto con il nome richiesto, o un riferimento nullo se l'oggetto non esiste

Argomenti

- name : QString
  Nome dell'oggetto.

Note

- Deve essere virtuale;
- Deve essere esplicitamente marcato come costante.

`+ void exportFile(path : QString, )`
Esporta il file dato il percorso

Argomenti

- path : QString
  Percorso dove esportare il file convertito.

Note

- Deve essere astratto.

`+ void importFile(path : QString, )`
Importa un file dato il percorso. È richiesto il caricamento del file grazie alla libreria esterna Assimp, e la creazione dei corretti SceneObject a partire dalle strutture dati di Assimp. È richiesto l'uso dei metodi privati di utilità buildObjects e buildLights per costruire i corretti oggetti di tipo Mesh e Light partendo dai dati restituiti da Assimp. È necessaria inoltre l'applicazione dei limiti di importazione della scena$_G$ grazie all'invocazione del metodo importLimits

Argomenti

- path : QString
  Percorso da dove importare il file da convertire.

Note

- Deve essere virtuale.

`# void signalUpdatedScene()`
Notifica l'avvenuto aggiornamento della scena$_G$
Note

- Deve essere virtuale.

`# void signalError()`
Manda un segnale per indicare che vi è stato un errore interno alla scena$_G$
Note

- Deve essere virtuale.

`+ QUndoStack& getCommandStack()`
Ritorna il riferimento all'oggetto contenente la lista delle operazioni effettuate
Note

- Deve essere virtuale;
- Deve essere esplicitamente marcato come costante.

`+ QStringList getObjectNames()`
Restituisce una lista con i nomi degli oggetti presenti nella scena$_G$
Note

- Deve essere virtuale.

`+ SceneObject getSelectedObject()`
Ritorna l'oggetto selezionato
Note

- Deve essere esplicitamente marcato come costante.

`+ void setSelectObject(selectedObject : SceneObject, )`
Imposta l'oggetto selezionato
Argomenti

- selectedObject : SceneObject
  Oggetto da selezionare.

`+ QUndoStack getCommandStack()`
Ritorna lo stack dei commandi

`+ void selectByName(name : QString, )`
Imposta come selezionato l'oggetto con il nome specificato, o non modifica nulla se l'oggetto non è presente nella scena$_G$
Argomenti

- name : QString
  Nome dell'oggetto da selezionare.

Note

- Deve essere astratto.

`+ T* getObjectType(name : QString, )`
Ritorna il tipo dell'oggetto
Argomenti

- name : QString
  Nome dell'oggetto da controllare.

Note

- Deve essere esplicitamente marcato come costante.

`+ void update()`
Emette il signal$_G$ di update

`# void selectionChanged()`
Invia un segnale quando viene selezionato un nuovo oggetto

`# void fileLoaded()`
Emette un segnale quando è stato caricato un file

`+ QMap<QString, SceneObject *>& getAllObjects()`
Ritorna una mappa associativa con nome e riferimento all'oggetto per tutti gli oggetti della scena$_G$
Note

- Deve essere astratto;
- Deve essere esplicitamente marcato come costante.

`+ void addObject(name : QString, object : SceneObject*, )`
Aggiunge un oggetto alla scena$_G$
Argomenti

- name : QString
  Nome da associare all'oggetto da aggiungere alla scena$_G$;
- object : SceneObject*
  Riferimento all'oggetto da aggiungere alla scena$_G$.

Note

- Deve essere astratto.

`+ int getLightNumber()`
Restituisce il numero di luci nella scena$_G$
Note

- Deve essere astratto;
- Deve essere esplicitamente marcato come costante.

`+ void removeObject(name : QString, )`
Rimuove un oggetto dalla scena$_G$ senza cancellarlo
Argomenti

- name : QString
  Nome dell'oggetto della scena$_G$ da eliminare.

Note

- Deve essere astratto.

3.7.2 SceneAdapter (class)

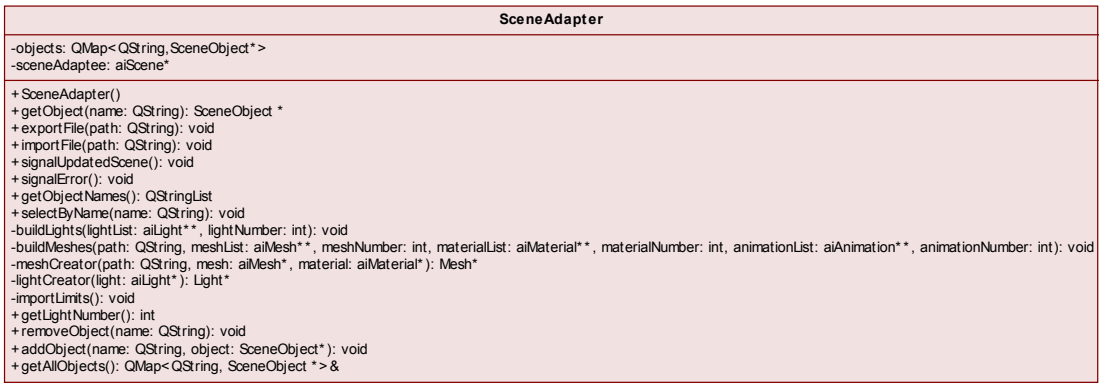

Figura 34: Classe SceneAdapter

Descrizione

Classe che viene utilizzata come adattatore per la libreria esterna Assimp. Rappresenta il componente adapter del Design Pattern$_G$ Adapter;

Utilizzo

Viene utilizzata come adattatore tra quanto esposto dalla libreria esterna Assimp e la classe Scene;

Classi ereditate

- DDDMob :: Model :: CConverter :: CLoaderModel :: Scene.

Attributi

`- QMap<QString,SceneObject*> objects`

Mappa tra i nomi e i riferimenti agli oggetti contenuti nella scena$_G$;

`- aiScene* sceneAdaptee`

Riferimento alla struttura dati di Assimp che rappresenta la scena$_G$ da adattare.

Metodi

`+ SceneAdapter()`

Costruttore della classe SceneAdapter. Inizializza una mappa vuota per il campo dati objects

`+ SceneObject * getObject(name : QString, )`

Ritorna il riferimento all'oggetto con il nome richiesto, o un riferimento nullo se l'oggetto non esiste

Argomenti

- name : QString
  Nome dell'oggetto da cercare.

Note

- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ void exportFile(path : QString, )`

Dato il percorso, invoca il metodo statico di SceneExporter per la scelta dell'algoritmo di esportazione, fornendo un riferimento alla mappa degli oggetti della scena$_G$. Ottiene il salvataggio della scena$_G$ nel percorso specificato, con il formato scelto dall'utente

Argomenti

- path : QString
  Percorso dove esportare il file convertito.

Note

- Questo metodo è stato ridefinito.

`+ void importFile(path : QString, )`

Importa un file dato il percorso. È richiesto il caricamento del file grazie alla libreria esterna Assimp, e la creazione dei corretti SceneObject a partire dalle strutture dati di Assimp. È richiesto l'uso dei metodi privati di utilità buildObjects e buildLights per costruire i corretti oggetti di tipo Mesh e Light partendo dai dati restituiti da Assimp. È necessaria inoltre l'applicazione dei limiti di importazione della scena$_G$ grazie all'invocazione del metodo importLimits

Argomenti

- path : QString
  Percorso da dove importare il file da convertire.

Note

- Questo metodo è stato ridefinito.

`# void signalUpdatedScene()`

Notifica l'avvenuto aggiornamento della scena$_G$

Note

- Questo metodo è stato ridefinito.

`# void signalError()`

Notifica un errore interno alla scena$_G$

Note

- Questo metodo è stato ridefinito.

`+ QStringList getObjectNames()`

Restituisce una lista con i nomi degli oggetti presenti nella scena$_G$

Note

- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ void selectByName(name : QString, )`

Imposta come selezionato l'oggetto con il nome specificato, o non modifica nulla se l'oggetto non è presente nella scena$_G$

Argomenti

- name : QString
  Nome dell'oggetto da selezionare.

Note

- Questo metodo è stato ridefinito.

`- void buildLights(lightList : aiLight**, lightNumber : int, )`

Costruisce tutti gli oggetti di tipo luce grazie al metodo di utilità lightCreator, applicando anche i limiti di importazione ad esse associati

Argomenti

- lightList : aiLight**
  Array di riferimenti delle luci della scena$_G$;
- lightNumber : int
  Lunghezza dell'array contente le luci nella scena$_G$.

`- void buildMeshes(path : QString, meshList : aiMesh**, meshNumber : int, materialList : aiMaterial**, materialNumber : int, animationList : aiAnimation**, animationNumber : int, )`

Costruisce tutte le Mesh presenti nella scena$_G$ importata, grazie anche al metodo di utilità meshCreator, che crea un singolo oggetto Mesh

Argomenti

- path : QString
  La directory contenente il file che è stato caricato nella scena$_G$;
- meshList : aiMesh**
  Array di riferimenti alle mesh della scena$_G$;
- meshNumber : int
  Lunghezza dell'array contente le mesh della scena$_G$;
- materialList : aiMaterial**
  Array di riferimenti ai materiali delle mesh della scena$_G$;
- materialNumber : int
  Lunghezza dell'array contente i materiali;
- animationList : aiAnimation**
  Array di riferimenti alle animazioni della scena$_G$;
- animationNumber : int
  Lunghezza dell'array contente le animazioni.

`- Mesh* meshCreator(path : QString, mesh : aiMesh*, material : aiMaterial*, )`

Data una luce nella struttura dati di Assimp, crea un oggetto Mesh e ne imposta i parametri corretti. Aggiunge poi il riferimento alla mesh creata nella mappa di oggetti della scena$_G$

Argomenti

- path : QString
  La directory contenente il file che è stato caricato nella scena$_G$;
- mesh : aiMesh*
  Riferimento alla mesh;
- material : aiMaterial*
  Rifermento al materiale da applicare alla mesh.

`- Light* lightCreator(light : aiLight*, )`

Data una luce nella struttura dati di Assimp, crea un oggetto Light e ne imposta i parametri corretti. Aggiunge poi il riferimento alla luce creata nella mappa di oggetti della scena$_G$

Argomenti

- light : aiLight*
  Riferimento alla luce da aggiungere alla scena$_G$.

`- void importLimits()`

Applica i limiti di importazione riguardanti le texture ad ogni oggetto presente nella scena$_G$. È richiesto il ridimensionamento di ogni texture che ecceda i limiti massimi di dimensione in modo che vengano mantenute le proporzioni originarie e l'istogramma dei colori, il suo salvataggio in un nuovo file immagine che non sovrascriva il file precedente, e l'impostazione della nuova texture nella Mesh oggetto dell'applicazione dei limiti.

`+ int getLightNumber()`

Restituisce il numero di luci presenti nella scena$_G$

Note

- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ void removeObject(name : QString, )`

Rimuove un oggetto dalla scena$_G$ senza cancellarlo

Argomenti

- name : QString
  Nome dell'oggetto della scena$_G$ da eliminare.

Note

- Questo metodo è stato ridefinito.

`+ void addObject(name : QString, object : SceneObject*, )`

Aggiunge il riferimento dell'oggetto alla mappa di oggetti della scena$_G$ con il nome passato come parametro

Argomenti

- name : QString
  Nome da associare all'oggetto da aggiungere alla scena$_G$;
- object : SceneObject*
  Riferimento all'oggetto da aggiungere alla scena$_G$.

Note

- Questo metodo è stato ridefinito.

`+ QMap<QString, SceneObject *>& getAllObjects()`

Ritorna una mappa associativa con nome e riferimento all'oggetto per tutti gli oggetti della scena$_G$

Note

- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

3.7.3 ImporterJson (class)

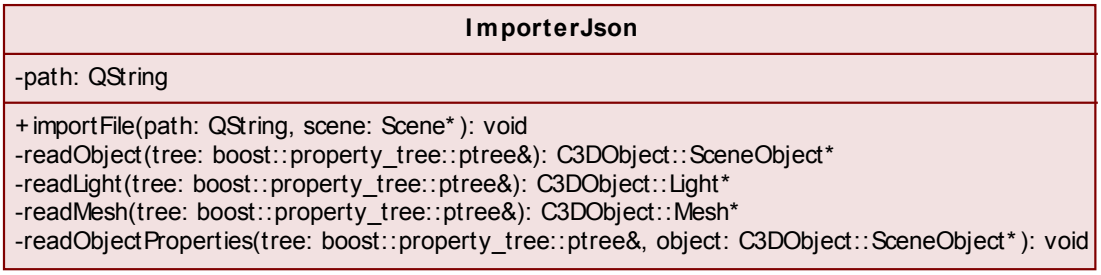

Figura 35: Classe ImporterJson

Descrizione

Classe che rappresenta un algoritmo di importazione della scena$_G$ in un file di formato JSON$_G$;

Utilizzo

Viene utilizzata per creare tutti gli oggetti della scena$_G$ a partire da un file JSON$_G$;

Attributi

`- QString path`

Percorso del file da importare.

Metodi

`+ void importFile (path : QString, scene : Scene*, )`

Metodo che permette l'importazione del file nel percorso specificato nel parametro. Deve leggere il file ed estrarre le informazioni degli oggetti contenute all'interno, utilizzando anche i metodi privati di utilità definiti per creare i tipi di oggetto popolati correttamente

Argomenti

- path : QString
  Percorso del file da importare;
- scene : Scene*
  Scena$_G$ da popolare.

`- C3DObject::SceneObject* readObject(tree : boost::property_tree::ptree& , )`

Legge uno SceneObject dal file

Argomenti

- tree : boost::property_tree::ptree&
  Albero da cui estrarre lo SceneObject.

`- C3DObject::Light* readLight(tree : boost::property_tree::ptree& , )`

Legge una Light dal file

Argomenti

- tree : boost::property_tree::ptree&
  Albero da cui estrarre la Light.

`- C3DObject::Mesh* readMesh(tree : boost::property_tree::ptree& , )`
Legge una Mesh dal file
Argomenti

- tree : boost::property_tree::ptree&
  Albero da cui estrarre la Mesh.

`- void readObjectProperties(tree : boost::property_tree::ptree& , object : C3DObject::SceneObject* , )`
Legge le proprietà di un oggetto
Argomenti

- tree : boost::property_tree::ptree&
  Albero da cui estrarre le proprietà;
- object : C3DObject::SceneObject*
  Oggetto da popolare.

3.8 DDDMob::Controller

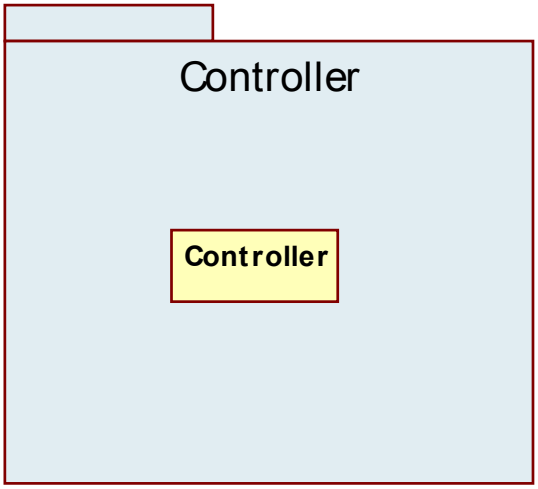

Figura 36: Componente DDDMob::Controller

Package$_G$ per il componente Controller dell'architettura MVC

3.8.1 Controller (class)

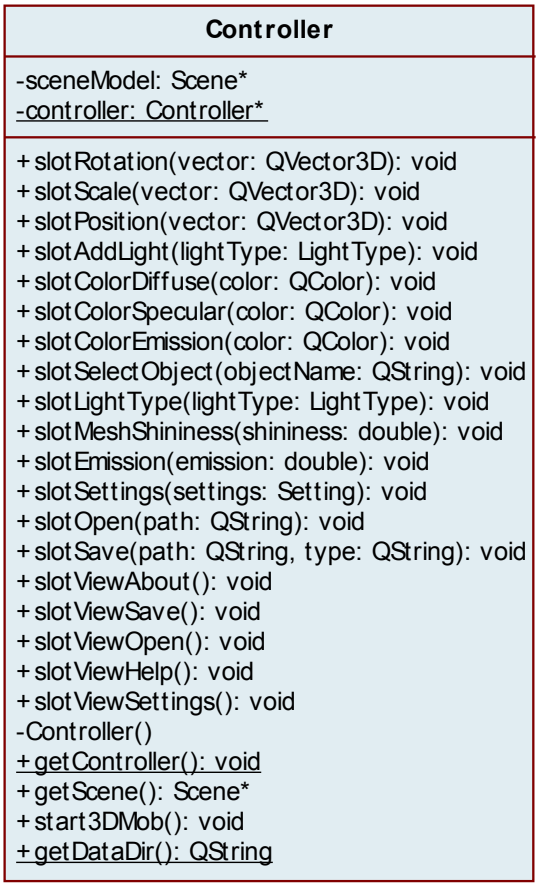

Figura 37: Classe Controller

Descrizione

Classe che rappresenta il componente controller del Design Pattern$_G$ MVC ed il componente client del Design Pattern$_G$ Command;

Utilizzo

Gestisce i Signal$_G$ inviati dalla View ed agisce nel modo corretto. Genera i comandi da eseguire sulla scena$_G$ 3D e memorizza tali comandi nel componente Scene del Model. Concorre nell'applicare le modifiche all'oggetto selezionato della scena$_G$ 3D;

Attributi

`- Scene* sceneModel`
Riferimento alla scena$_G$ del modello;

`- Controller* controller`
Riferimento a se stesso per l'accesso protetto.

Metodi

`+ void slotRotation(vector : QVector3D, )`
riceve la richiesta di rotazione
Argomenti

- vector : QVector3D
  Vettore che indica di quanto ruotare l'oggetto.

`+ void slotScale(vector : QVector3D, )`
riceve la richiesta di rotazione
Argomenti

- vector : QVector3D
  Vettore che indica di quanto ridimensionare l'oggetto.

`+ void slotPosition(vector : QVector3D, )`
Riceve la richiesta di spostamento
Argomenti

- vector : QVector3D
  Vettore che indica di quanto translare l'oggetto.

`+ void slotAddLight(lightType : LightType, )`
riceve la richiesta di aggiunta luce
Argomenti

- lightType : LightType
  Tipo di luce.

`+ void slotColorDiffuse(color : QColor, )`
riceve la richiesta di cambio colore di diffusione$_G$
Argomenti

- color : QColor
  Colore di diffusione$_G$ che si desidera venga impostato sull'oggetto selezionato.

`+ void slotColorSpecular(color : QColor, )`
riceve la richiesta di modifica del colore speculare$_G$
Argomenti

- color : QColor
  Colore speculare$_G$ che si desidera venga impostato sull'oggetto selezionato.

`+ void slotColorEmission(color : QColor, )`
riceve la richiesta di modifica del colore di emissione
Argomenti

- color : QColor
  Colore di emissione che si desidera venga impostato sull'oggetto selezionato.

`+ void slotSelectObject(objectName : QString, )`
riceve la richiesta di modifica dell'oggetto selezionato
Argomenti

- objectName : QString
  Nome dell'oggetto da selezionare.

`+ void slotLightType(lightType : LightType, )`
riceve la richiesta di modifica del tipo della luce
Argomenti

- lightType : LightType
  Tipo di luce.

`+ void slotMeshShininess(shininess : double, )`
riceve la richiesta di modifica della lucentezza del mesh
Argomenti

- shininess : double
  Lucentezza della mesh.

`+ void slotEmission(emission : double, )`
riceve la richiesta di modifica dell'emissione dell'oggetto
Argomenti

- emission : double
  Il valore di emissività che si vuole sia impostato sull'oggetto della scena$_G$ selezionato.

`+ void slotSettings(settings : Setting, )`
Riceve richieste sulle impostazioni del sistema
Argomenti

- settings : Setting
  Impostazioni.

`+ void slotOpen(path : QString, )`
Riceve la richiesta di apertura di un file
Argomenti

- path : QString
  Percorso del file.

`+ void slotSave(path : QString, type : QString, )`
Riceve la richiesta di salvataggio della scena$_G$ su file
Argomenti

- path : QString
  Percorso del file;
- type : QString
  Tipo del file da salvare.

`+ void slotViewAbout()`
Riceve il segnale per visualizzare le informazioni sul programma

`+ void slotViewSave()`
Richiede l'apertura della finestra di salvataggio

`+ void slotViewOpen()`
Richiede l'apertura della finestra di apertura file

`+ void slotViewHelp()`
Richiede l'apertura della finestra di aiuto

`+ void slotViewSettings()`
Richiede l'apertura della finestra di impostazione delle preferenze del programma

`- Controller()`
Costruttore privato della classe Controller

`+ void getController()`
Restituisce un riferimento a se stesso in modo protetto
Note

- Deve essere un metodo statico.

`+ Scene* getScene()`
Ritorna il puntatore alla scena$_G$ corrente
Note

- Deve essere esplicitamente marcato come costante.

`+ void start3DMob()`
Avvia la finestra principale dell'applicazione

`+ QString getDataDir()`
Ritorna la cartella contenente i dati del programma
Note

- Deve essere un metodo statico.

3.9 DDDMob::View

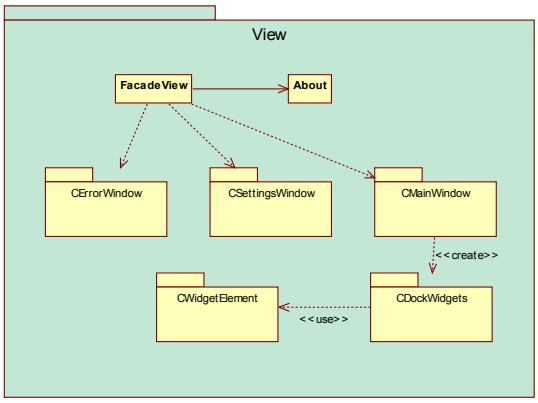

Figura 38: Componente DDDMob::View

Package$_G$ per il componente View dell'architettura MVC

3.9.1 FacadeView (class)

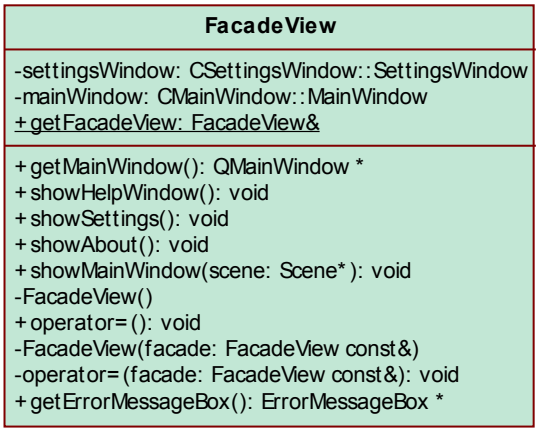

Figura 39: Classe FacadeView

Descrizione

Classe che rappresenta la Vista. È la componente facade del Design Pattern$_G$ Facade, ed è implementata con il Design Pattern$_G$ Singleton;

Utilizzo

Viene utilizzata per accedere alla vista in modo protetto, in quanto espone i metodi necessari per l'interazione dall'esterno, nascondendo l'implementazione e la struttura;

Attributi

`- CSettingsWindow::SettingsWindow settingsWindow`
Riferimento all'interfaccia per le impostazioni del programma;

`- CMainWindow::MainWindow mainWindow`
Riferimento alla finestra principale;

`+ FacadeView& getFacadeView`
Ritorna il riferimento protetto alla Vista.

Metodi

`+ QMainWindow * getMainWindow()`
Ritorna il riferimento protetto alla finestra principale
Note
- Deve essere esplicitamente marcato come costante.

`+ void showHelpWindow()`
Mostra la finestra che permette l'accesso al sistema di aiuto

`+ void showSettings()`
Mostra la finestra con le informazioni sulle impostazioni del programma

`+ void showAbout()`
Mostra la finestra con le informazioni sul programma

`+ void showMainWindow(scene : Scene*, )`
Mostra la finestra principale del programma
Argomenti

- scene : Scene*
  Scena$_G$ da mostrare nella finestra principale.

`- FacadeView()`
Costruttore della classe

`+ void operator=()`
Ridefinizione dell'operatore di assegnazione

`- FacadeView(facade : FacadeView const&, )`
Costrutture della classe
Argomenti

- facade : FacadeView const&
  Riferimento all'oggetto FacadeView.

`- void operator=(facade : FacadeView const&, )`
Ridefinizione dell'operatore =
Argomenti

- facade : FacadeView const&
  Riferimento all'oggetto FacadeView.

`+ ErrorMessageBox * getErrorMessageBox()`
Ritorna un riferimento a una finestra di messaggio di errore
Note

- Deve essere esplicitamente marcato come costante.

3.9.2 About (class)

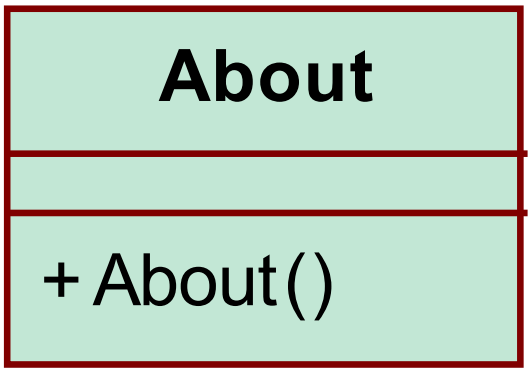

Figura 40: Classe About

Descrizione
Classe che rappresenta la finestra contenente le informazioni sul programma e sui suoi creatori;

Utilizzo

Viene utilizzata per visualizzare le informazioni sul programma e sui suoi creatori, oltre che la versione del prodotto e delle librerie utilizzate;

Classi ereditate

- QDialog.

Metodi

`+ About()`
Costruttore di About

3.10 DDDMob::View::CDockWidgets

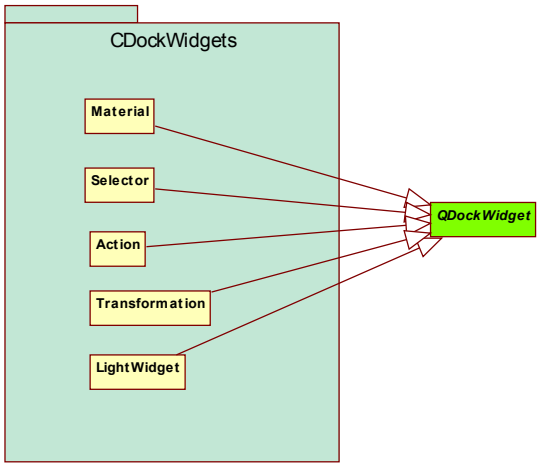

Figura 41: Componente DDDMob::View::CDockWidgets

Componente che contiene tutti i tipi di dockwidget disponibili nel sistema

3.10.1 Material (class)

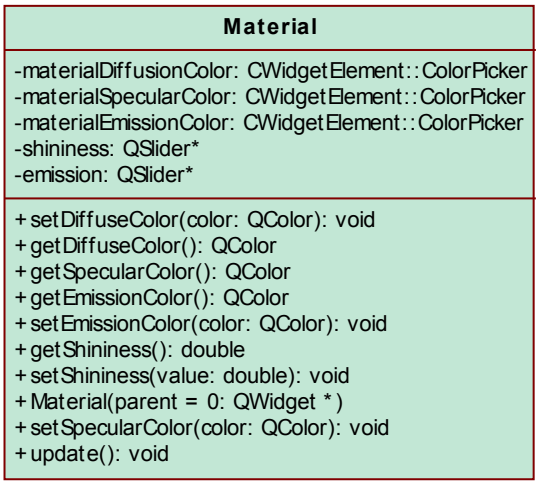

Figura 42: Classe Material

Descrizione

Classe che rappresenta il widget contenente le informazioni sul materiale;

Utilizzo

Viene utilizzata per permettere di cambiare le caratteristiche del materiale dell'oggetto. Emette un Signal$_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Attributi

`- CWidgetElement::ColorPicker materialDiffusionColor`
Elemento grafico che permette di modificare il colore di diffusione$_G$ del materiale;

`- CWidgetElement::ColorPicker materialSpecularColor`
Elemento grafico che permette di modificare il colore speculare$_G$ del materiale;

`- CWidgetElement::ColorPicker materialEmissionColor`
Elemento grafico che permette di modificare il colore di emissione del materiale;

`- QSlider* shininess`
Elemento grafico che permette di modificare il valore della luminosità del materiale;

`- QSlider* emission`
Elemento grafico che permette di modificare il valore di emissione del materiale.

Metodi

`+ void setDiffuseColor(color : QColor, )`
Imposta il colore di diffusione$_G$
Argomenti

- color : QColor
  Colore da impostare.

`+ QColor getDiffuseColor()`
Ritorna il colore di diffusione$_G$ selezionato
Note

- Deve essere esplicitamente marcato come costante.

`+ QColor getSpecularColor()`
Ritorna il colore speculare$_G$ selezionato
Note

- Deve essere esplicitamente marcato come costante.

`+ QColor getEmissionColor()`
Ritorna il colore di emissione selezionato
Note

- Deve essere esplicitamente marcato come costante.

`+ void setEmissionColor(color : QColor, )`
Imposta il colore di emissione
Argomenti

- color : QColor
  Colore di emissione da impostare.

`+ double getShininess()`
Ritorna il valore di lucentezza del materiale
Note

- Deve essere esplicitamente marcato come costante.

`+ void setShininess(value : double , )`
Imposta il valore di lucentezza
Argomenti

- value : double
  Lucentezza da impostare.

`+ Material(parent = 0 : QWidget * , )`
Costruttore per la classe Material
Argomenti

- parent = 0 : QWidget *
  Puntatore al QWidget padre del Material.

`+ void setSpecularColor (color : QColor , )`
Imposta il colore speculare$_G$
Argomenti

- color : QColor
  Colore speculare$_G$ da impostare.

`+ void update()`
Metodo per aggiornare il materiale

3.10.2 Selector (class)

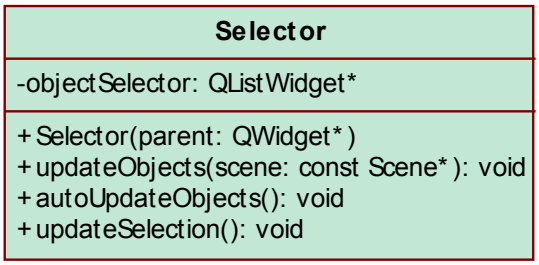

Figura 43: Classe Selector

Descrizione

Classe che rappresenta il widget contenente il selettore di oggetti;

Utilizzo

Viene utilizzata per selezionare un oggetto della scena$_G$. Emette un Signal$_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Attributi

`- QListWidget* objectSelector`

Elemento grafico che permette di selezionare un oggetto.

Metodi

`+ Selector(parent : QWidget*, )`

Costruttore della classe Selector

Argomenti

- parent : QWidget*
  Puntatore al QWidget padre di Selector.

`+ void updateObjects(scene : const Scene*, )`

Metodo che aggiorna la lista degli oggetti presenti nella scena$_G$

Argomenti

- scene : const Scene*
  Scena$_G$ da aggiornare.

`+ void autoUpdateObjects()`

Metodo che aggiorna automaticamente gli elementi presenti nella scena$_G$

`+ void updateSelection()`

Metodo che recupera l'elemento selezionato nella scena$_G$ e lo aggiorna nella lista di oggetti

3.10.3 LightWidget (class)

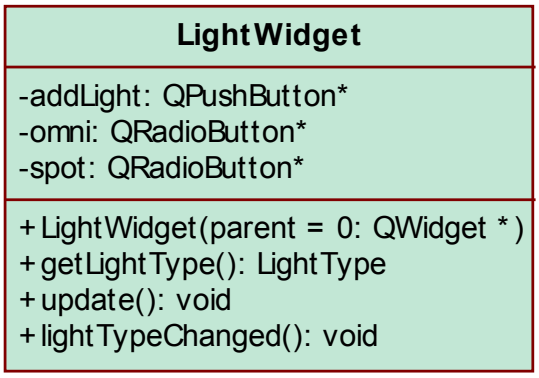

Figura 44: Classe LightWidget

Descrizione

Classe che rappresenta il widget che permette di interagire con le luci della scena$_G$ e di aggiungerne di nuove;

Utilizzo

Viene utilizzata per interagire con le luci della scena$_G$ o aggiungerne una nuova. Emette un Signal$_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Attributi

`- QPushButton* addLight`
Elemento grafico che permette di aggiungere una luce;

`- QRadioButton* omni`
Oggetto grafico per gestire la tipologia della luce;

`- QRadioButton* spot`
Elemento grafico per la gestione della luce di tipo spot.

Metodi

`+ LightWidget(parent = 0 : QWidget * , )`
Costruttore della classe LightWidget
Argomenti

- parent = 0 : QWidget *
  Puntatore al QWidget padre del ColorPicker.

`+ LightType getLightType()`
Ritorna il tipo di luce
Note

- Deve essere esplicitamente marcato come costante.

`+ void update()`
Aggiorna il tipo di luce

`# void lightTypeChanged()`
Metodo eseguito al cambio del tipo di luce

3.10.4 Action (class)

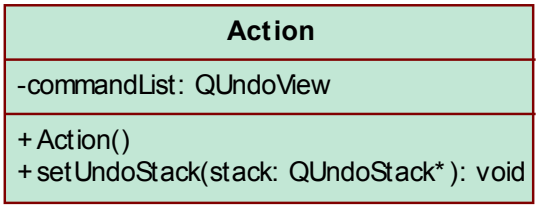

Figura 45: Classe Action

Descrizione
Classe che rappresenta il widget che consente di visualizzare le azioni effettuate, di annullarle e ripristinarle;

Utilizzo
Viene utilizzata per visualizzare le azioni effettuate, annullarle e ripristinarle. Emette un Signal_G in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Attributi

`- QUndoView commandList`
Elemento grafico che permette di visualizzare le operazioni effettuate.

Metodi

`+ Action()`
Costruttore della classe

`+ void setUndoStack(stack : QUndoStack*, )`
Imposta il range di comandi memorizzati
Argomenti

- stack : QUndoStack*
  QUndoStack da visualizzare in Action.

3.10.5 Transformation (class)

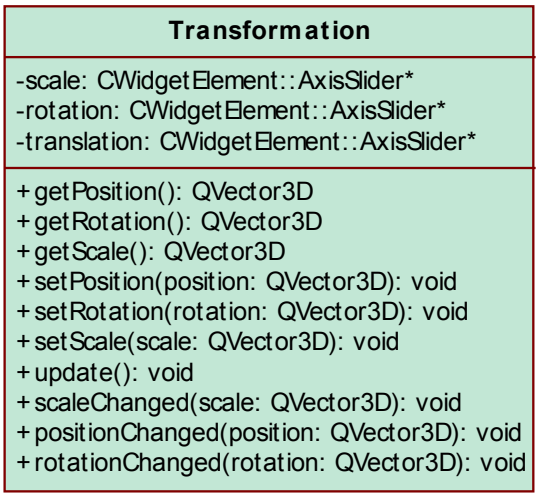

Figura 46: Classe Transformation

Descrizione

Classe che rappresenta un widget contenente gli slider per modificare alcune caratteristiche dell'oggetto;

Utilizzo

Viene utilizzata per permette di applicare una trasformazione alla scena$_G$. Emette un Signal$_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Attributi

`- CWidgetElement::AxisSlider* scale`
Elemento grafico che permette di modificare la dimensione;

`- CWidgetElement::AxisSlider* rotation`
Elemento grafico che permette di modificare la rotazione dell'elemento;

`- CWidgetElement::AxisSlider* translation`
Elemento grafico che permette di modificare la traslazione dell'elemento.

Metodi

`+ QVector3D getPosition()`
Ritorna il vettore rappresentante la posizione indicata
Note

- Deve essere esplicitamente marcato come costante.

`+ QVector3D getRotation()`
Ritorna il vettore rappresentante la rotazione indicata
Note

- Deve essere esplicitamente marcato come costante.

`+ QVector3D getScale()`

Ritorna il vettore rappresentante la dimensione indicata

Note

- Deve essere esplicitamente marcato come costante.

`+ void setPosition(position : QVector3D, )`

Imposta i valori visualizzati riguardanti la posizione in base al vettore passato

Argomenti

- position : QVector3D
  Vettore 3D, i componenti x, y, z di tale vettore impostano i valori degli slider.

`+ void setRotation(rotation : QVector3D, )`

Imposta i valori visualizzati riguardanti la rotazione in base al vettore passato

Argomenti

- rotation : QVector3D
  Vettore 3D, i componenti x, y, z di tale vettore impostano i valori degli slider.

`+ void setScale(scale : QVector3D, )`

Imposta i valori visualizzati riguardanti la dimensione in base al vettore passato

Argomenti

- scale : QVector3D
  Vettore 3D, i componenti x, y, z di tale vettore impostano i valori degli slider.

`+ void update()`

Aggiorna gli AxisSlider a seconda del modello

`# void scaleChanged(scale : QVector3D, )`

E' stata cambiata la scala

Argomenti

- scale : QVector3D
  Nuovo valore assunto dallo slider per il ridimensionamento.

`# void positionChanged(position : QVector3D, )`

E' cambiata la posizione

Argomenti

- position : QVector3D
  Nuovo valore assunto dallo slider per la posizione.

`# void rotationChanged(rotation : QVector3D, )`

E' cambiata la rotazione

Argomenti

- rotation : QVector3D
  Nuovo valore assunto dallo slider per la rotazione.

3.11 DDDMob::View::CMainWindow

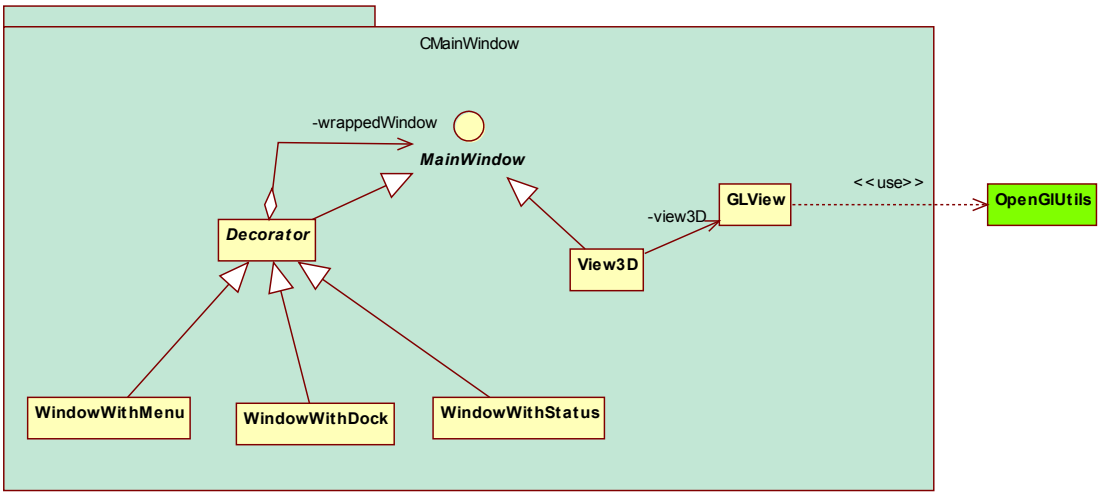

Figura 47: Componente DDDMob::View::CMainWindow

Componente parte di View per la finestra principale dell'applicazione

3.11.1 WindowWithMenu (class)

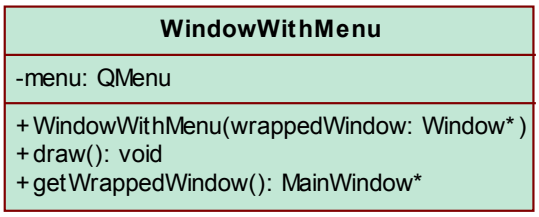

Figura 48: Classe WindowWithMenu

Descrizione

Classe che decora la finestra principale. Rappresenta uno dei componenti concrete decorator del Design Pattern$_G$ Decorator;

Utilizzo

Viene utilizzata per aggiungere un menù alla finestra;

Classi ereditate

- QMenuBar;
- DDDMob :: View :: CMainWindow :: Decorator.

Attributi

`- QMenu menu`

Menù nella barra dei menù.

Metodi

`+ WindowWithMenu(wrappedWindow : Window*, )`
Costruttore della classe WindowWithMenu
Argomenti

- wrappedWindow : Window*
  La composizione di finestre decorate che si vuole decorare con la classe.

`+ void draw()`
Metodo per rendere visibile la finestra
Note

- Questo metodo è stato ridefinito.

`+ MainWindow* getWrappedWindow()`
Metodo che ritorna un riferimento alla View3D
Note

- Questo metodo è stato ridefinito.

3.11.2 WindowWithDock (class)

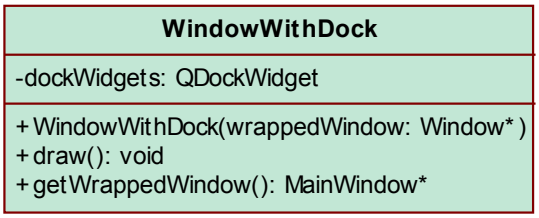

Figura 49: Classe WindowWithDock

Descrizione
Classe che decora la finestra principale. Rappresenta uno dei componenti concrete decorator del Design Pattern$_G$ Decorator;

Utilizzo
Decoratore che aggiunge elementi ancorabili alla finestra;

Classi ereditate

- DDDMob :: View :: CMainWindow :: Decorator.

Attributi

`- QDockWidget dockWidgets`
Rappresenta i dockWidgets presenti nella finestra.

Metodi

`+ WindowWithDock(wrappedWindow : Window*, )`
Costruttore della classe WindowWithDock
Argomenti

- wrappedWindow : Window*
  La composizione di finestre che si vuole decorare con la classe.

`+ void draw()`
Metodo che disegna la finestra applicando i dockWidgets
Note
- Questo metodo è stato ridefinito.

`+ MainWindow* getWrappedWindow()`
Metodo che ritorna la View3D
Note
- Questo metodo è stato ridefinito.

3.11.3 WindowWithStatus (class)

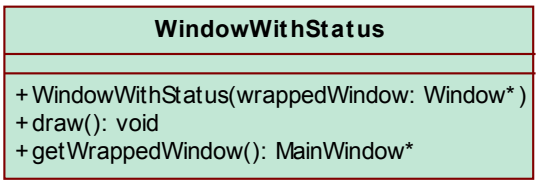

Figura 50: Classe WindowWithStatus

Descrizione
Classe che decora la finestra principale. Rappresenta uno dei componenti concrete decorator del Design Pattern$_G$ Decorator;

Utilizzo
Viene utilizzata per aggiungere ad una finestra la barra di stato;

Classi ereditate

- QStatusBar;
- DDDMob :: View :: CMainWindow :: Decorator.

Metodi

`+ WindowWithStatus(wrappedWindow : Window*, )`
Costruttore della classe WindowWithStatus
Argomenti
- wrappedWindow : Window*
  La composizione di finestre che si vuole decorare con la classe.

`+ void draw()`
Metodo per rendere visibile la finestra
Note
- Questo metodo è stato ridefinito.

`+ MainWindow* getWrappedWindow()`
Metodo che ritorna un riferimento alla View3D
Note
- Questo metodo è stato ridefinito.

3.11.4 Decorator (abstract)

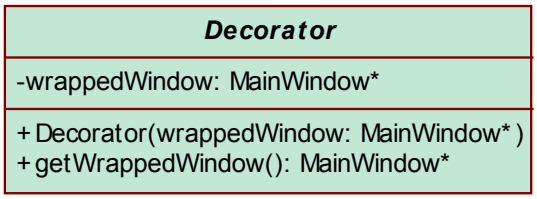

Figura 51: Classe Decorator

Descrizione

Classe astratta che rappresenta le varie decorazioni applicabili all'interfaccia utente. Rappresenta il componente decorator del Design Pattern$_G$ Decorator;

Utilizzo

Classe che rappresenta le possibili decorazioni applicabili all'interfaccia utente, specificate poi dalle classi che ereditano da questa;

Classi ereditate

- DDDMob :: View :: CMainWindow :: MainWindow.

Ereditata da

- DDDMob :: View :: CMainWindow :: WindowWithMenu;
- DDDMob :: View :: CMainWindow :: WindowWithDock;
- DDDMob :: View :: CMainWindow :: WindowWithStatus.

Attributi

`- MainWindow* wrappedWindow`

Puntatore alla composizione di finestre che si vuole decorare con la classe.

Metodi

`+ Decorator(wrappedWindow : MainWindow*, )`

Costruttore della classe Decorator

Argomenti

- wrappedWindow : MainWindow*
  La composizione di finestre che si vuole decorare con la classe.

`+ MainWindow* getWrappedWindow()`

Metodo che ritorna la View3D

Note

- Deve essere virtuale;
- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

3.11.5 MainWindow (interface)

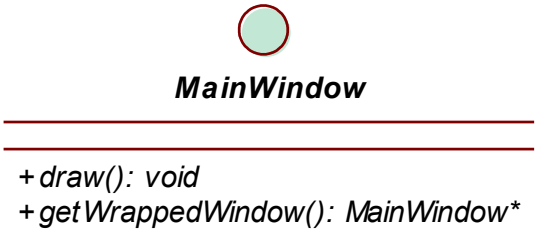

Figura 52: Classe MainWindow

Descrizione

Interfaccia che rappresenta la finestra principale del programma, ottenuta dalla vista 3D decorata con le decorazioni offerte dalla classe Decorator. Rappresenta il componente component del Design Pattern$_G$ Decorator;

Utilizzo

Ereditata da

- DDDMob :: View :: CMainWindow :: Decorator;
- DDDMob :: View :: CMainWindow :: View3D.

Metodi

`+ void draw()`

Metodo per rendere visibile la finestra

Note

- Deve essere astratto.

`+ MainWindow* getWrappedWindow()`

Metodo che ritorna un riferimento alla View3D

Note

- Deve essere astratto.

3.11.6 View3D (class)

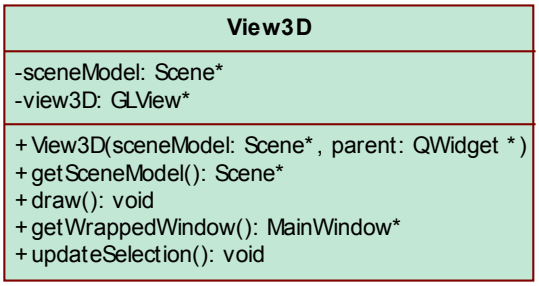

Figura 53: Classe View3D

Descrizione

Classe che rappresenta la vista contenente la scena$_G$ 3D. È il componente concrete component del Design Pattern$_G$ Decorator;

Utilizzo

Viene utilizzata per visualizzare l'anteprima della scena$_G$ 3D del Model. Manda Signal$_G$ di richiesta di aggiornamento dei dati di visualizzazione della scena$_G$ 3D al Model. Può leggere i dati aggiornati direttamente dal Model dopo aver ricevuto un Signal$_G$ di aggiornamento;

Classi ereditate

- QMainWindow;
- DDDMob :: View :: CMainWindow :: MainWindow.

Attributi

`- Scene* sceneModel`
Riferimento alla scena$_G$ del Modello;

`- GLView* view3D`
Finestra di visualizzazione della scena$_G$ 3D.

Metodi

`+ View3D(sceneModel : Scene*, parent : QWidget *, )`
Costruttore di View3D
Argomenti

- sceneModel : Scene*
  Riferimento alla scena$_G$ da creare;
- parent : QWidget *
  Indica il padre dell'elemento. Se il valore è 0 l'elemento diverrà una nuova finestra. Se tale valore è definito sarà parte del padre passato come parametro. Quando il padre viene cancellato, verrà cancellato anche il figlio.

`+ Scene* getSceneModel()`
Ritorna un riferimento costante alla Scena$_G$
Note

- Deve essere esplicitamente marcato come costante.

`+ void draw()`
Metodo per rendere visibile la finestra
Note

- Questo metodo è stato ridefinito.

`+ MainWindow* getWrappedWindow()`
Metodo che ritorna un riferimento alla View3D
Note

- Questo metodo è stato ridefinito.

`+ void updateSelection()`
Aggiorna i dati relativi alla vista

3.11.7 GLView (class)

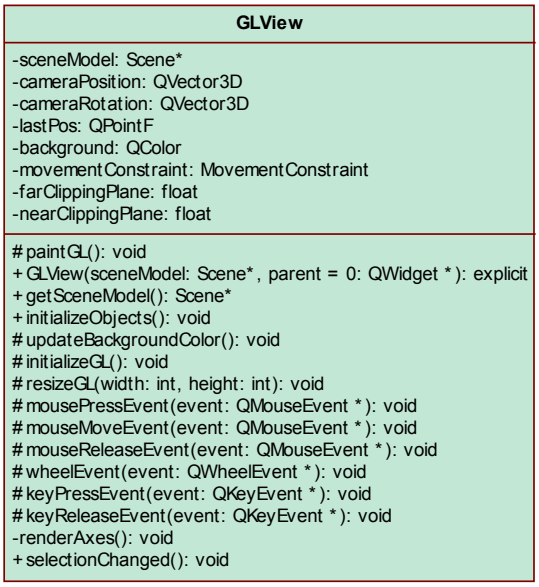

Figura 54: Classe GLView

Descrizione

Classe che esegue il rendering$_G$ della scena$_G$;

Utilizzo

Classi ereditate

- QGLWidget.

Attributi

`- Scene* sceneModel`
Modello della scena$_G$;

`- QVector3D cameraPosition`
Centro di rotazione della camera$_G$;

`- QVector3D cameraRotation`
Angoli di rotazione della camera$_G$;

`- QPointF lastPos`
Tracciamento della posizione del mouse;

`- QColor background`
Colore di sfondo;

`- MovementConstraint movementConstraint`
Tipologia di movimento;

`- float farClippingPlane`
Distanza del clipping plane per il tronco di visualizzazione;

`- float nearClippingPlane`
Distanza del clipping plane per il tronco di visualizzazione.

Metodi

`# void paintGL()`
Aggiorna la scena$_G$
Note
- Questo metodo è stato ridefinito.

`+ explicit GLView(sceneModel : Scene* , parent = 0 : QWidget *, )`
Costruttore di GLView
Argomenti
- sceneModel : Scene*
  Puntatore alla scena$_G$ da rappresentare;
- parent = 0 : QWidget *
  Puntatore al QWidget padre della GLView.

`+ Scene* getSceneModel()`
Ritorna il modello della scena$_G$
Note
- Deve essere esplicitamente marcato come costante.

`+ void initializeObjects()`
Inizializza gli oggetti della scena$_G$

`# void updateBackgroundColor()`
Aggiorna il colore di sfondo

`# void initializeGL()`
Inizializza OpenGL$_G$
Note
- Questo metodo è stato ridefinito.

`# void resizeGL(width : int, height : int, )`
Ridimensiona la scena$_G$
Argomenti
- width : int
  Nuova larghezza della GLView;
- height : int
  Nuova altezza della GLView.

Note
- Questo metodo è stato ridefinito.

`# void mousePressEvent(event : QMouseEvent *, )`
Metodo eseguito alla pressione di un tasto del mouse
Argomenti
- event : QMouseEvent *
  Oggetto contenente tutti i parametri sull'evento avvenuto.

Note
- Questo metodo è stato ridefinito.

`# void mouseMoveEvent(event : QMouseEvent *, )`
Metodo eseguito al movimento del mouse
Argomenti

- event : QMouseEvent *
  Oggetto contenente tutti i parametri sull'evento avvenuto.

Note

- Questo metodo è stato ridefinito.

`# void mouseReleaseEvent(event : QMouseEvent *, )`

Metodo eseguito al rilascio di un tasto premuto sul mouse

Argomenti

- event : QMouseEvent *
  Oggetto contenente tutti i parametri sull'evento avvenuto.

Note

- Questo metodo è stato ridefinito.

`# void wheelEvent(event : QWheelEvent *, )`

Metodo eseguito alla rotazione della rotellina del mouse

Argomenti

- event : QWheelEvent *
  Oggetto contenente tutti i parametri sull'evento avvenuto.

Note

- Questo metodo è stato ridefinito.

`# void keyPressEvent(event : QKeyEvent *, )`

Metodo eseguito alla pressione di un tasto della tastiera

Argomenti

- event : QKeyEvent *
  Oggetto contenente tutti i parametri sull'evento avvenuto.

Note

- Questo metodo è stato ridefinito.

`# void keyReleaseEvent(event : QKeyEvent *, )`

Metodo eseguito al rilascio di un tasto premuto sulla tastiera

Argomenti

- event : QKeyEvent *
  Oggetto contenente tutti i parametri sull'evento avvenuto.

Note

- Questo metodo è stato ridefinito.

`- void renderAxes()`

Esegue il render sugli assi

`# void selectionChanged()`

Cambio selezione

3.12 DDDMob::View::CSettingsWindow

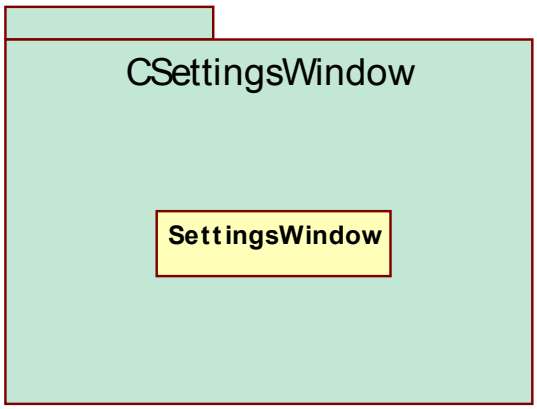

Figura 55: Componente DDDMob::View::CSettingsWindow

Componente parte di View per la finestra di configurazione delle impostazioni del programma

3.12.1 SettingsWindow (class)

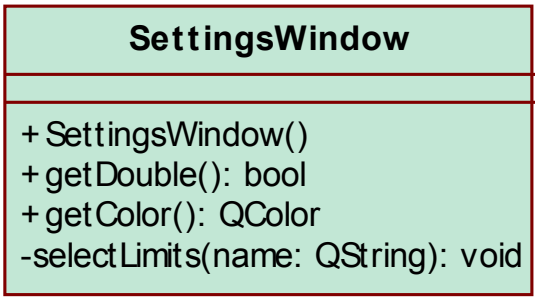

Figura 56: Classe SettingsWindow

Descrizione

Classe che rappresenta la finestra per modificare le impostazioni del programma;

Utilizzo

Viene utilizzata per visualizzare le impostazioni del programma. La sua apertura deriva dalla ricezione di un Signal_G da parte del Controller. Manda Signal_G di richiesta di aggiornamento dei dati di configurazione al Model. Può leggere i dati aggiornati direttamente dal Model dopo aver ricevuto un Signal_G di aggiornamento;

Classi ereditate

- QWindow.

Metodi

`+ SettingsWindow()`

Costruttore della classe

`+ bool getDouble()`
Ritorna il settaggio numerico dell'applicazione
Note
- Deve essere esplicitamente marcato come costante.

`+ QColor getColor()`
Ritorna il colore di sfondo
Note
- Deve essere esplicitamente marcato come costante.

`- void selectLimits(name : QString , )`
Seleziona un settaggio
Argomenti
- name : QString
  Nome del device da selezionare.

3.13 DDDMob::View::CErrorWindow

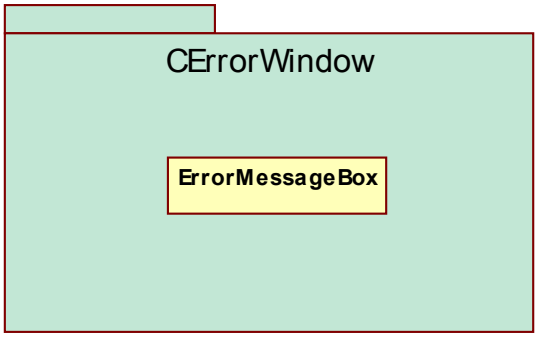

Figura 57: Componente DDDMob::View::CErrorWindow

Componente parte di View per la finestra di segnalazione degli errori

3.13.1 ErrorMessageBox (class)

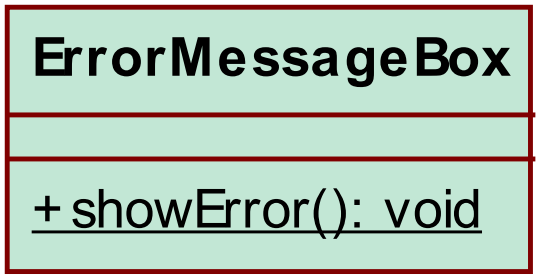

Figura 58: Classe ErrorMessageBox

Descrizione
Classe che rappresenta una finestra che riporta messaggi d'errore per l'utente;

Utilizzo

La sua apertura deriva dalla ricezione di un $Signal_G$ da parte del Model, generato nel caso in cui si verifichi una inconsistenza durante l'azione del Model. Il $Signal_G$ include un codice di errore e visualizza il relativo messaggio di errore;

Metodi

`+ void showError()`

Riceve il codice di errore e lo mostra

Note

- Deve essere un metodo statico.

3.14 DDDMob::View::CWidgetElement

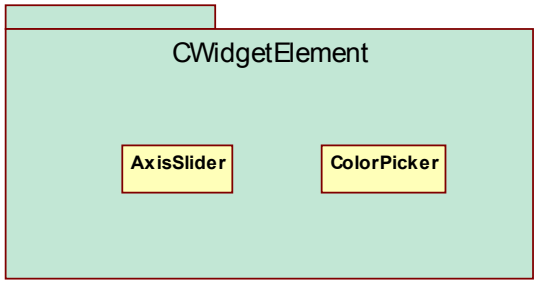

Figura 59: Componente DDDMob::View::CWidgetElement

Componente per elementi che sono utilizzati dai widget di View

3.14.1 AxisSlider (class)

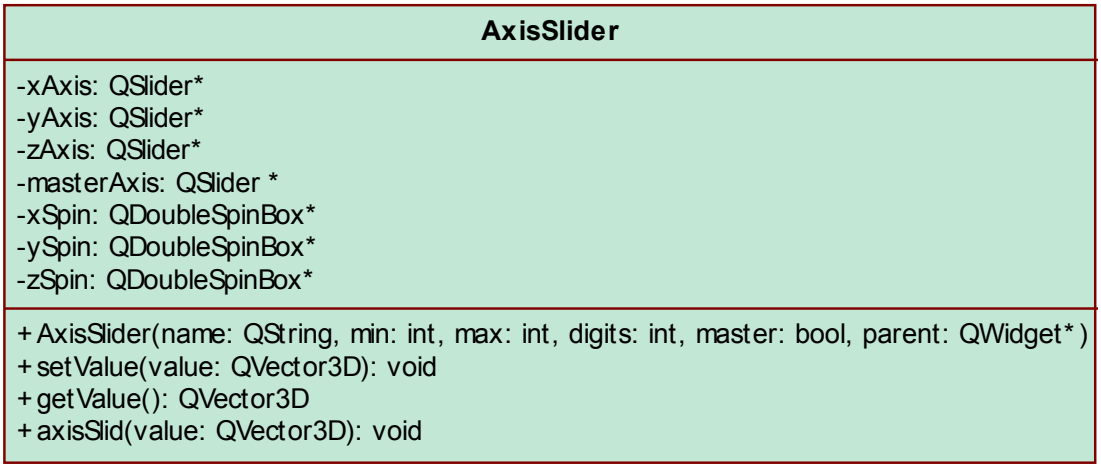

Figura 60: Classe AxisSlider

Descrizione

Classe che rappresenta il widget per modificare dei valori sugli assi x, y, z;

Utilizzo

Viene utilizzata per inviare il $signal_G$ opportuno al controller per la modifica dei valori;

Classi ereditate

- QGroupBox.

Attributi

`- QSlider* xAxis`
Slider che permette di cambiare le coordinate lungo l'asse x del mesh;

`- QSlider* yAxis`
Slider che permette di cambiare le coordinate lungo l'asse y del mesh;

`- QSlider* zAxis`
Slider che permette di cambiare le coordinate lungo l'asse z del mesh;

`- QSlider * masterAxis`
Slider che controlla gli altri;

`- QDoubleSpinBox* xSpin`
Elemento di input per controllare in modo preciso il valore di xAxis;

`- QDoubleSpinBox* ySpin`
Elemento di input per controllare in modo preciso il valore di yAxis;

`- QDoubleSpinBox* zSpin`
Elemento di input per controllare in modo preciso il valore di zAxis.

Metodi

`+ AxisSlider(name : QString, min : int, max : int, digits : int, master : bool, parent : QWidget*, )`
Costruttore di AxisSlider
Argomenti

- name : QString
  Testo da visualizzare per riconoscere il widget;
- min : int
  Minimo valore accettato;
- max : int
  Massimo valore accettato;
- digits : int
  Numero di cifre decimali da utilizzare;
- master : bool
  Se creare uno slider che controlla il valore degli altri tre;
- parent : QWidget*
  Puntatore al widget contenitore.

`+ void setValue(value : QVector3D, )`
Imposta i valori degli slider
Argomenti

- value : QVector3D
  Vettore con i valori x, y, z da impostare.

`+ QVector3D getValue()`
Ritorna un vettore contenente il valore rappresentato dagli slider
Note
- Deve essere esplicitamente marcato come costante.

`# void axisSlid(value : QVector3D, )`
Segnale emesso quando il valore degli slider cambia
Argomenti
- value : QVector3D
  Vettore contenente il valore rappresentato dagli slider.

3.14.2 ColorPicker (class)

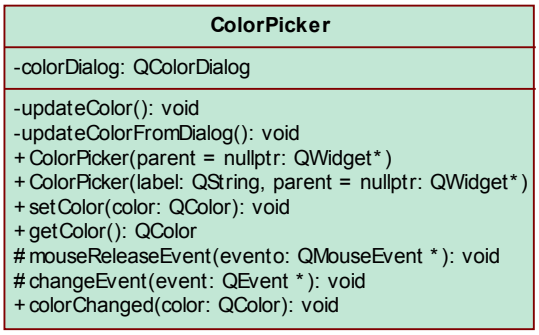

Figura 61: Classe ColorPicker

Descrizione
Classe che rappresenta il widget che consente di selezionare un colore;

Utilizzo
Viene utilizzata per permette di modificare il colore selezionato. Modifiche al colore selezionato provocano un signal$_G$ appropriato;

Classi ereditate
- QPushButton.

Attributi

`- QColorDialog colorDialog`
Form per la scelta del colore.

Metodi

`- void updateColor()`
Aggiorna il widget

`- void updateColorFromDialog()`
Aggiorna la finestra di dialogo

`+ ColorPicker(parent = nullptr : QWidget* , )`

Costruttuore dell'oggetto

Argomenti

- parent = nullptr : QWidget*
  Puntatore al QWidget padre del ColorPicker.

`+ ColorPicker(label : QString , parent = nullptr : QWidget* , )`

Costruttore dell'oggetto

Argomenti

- label : QString
  Stringa visualizzata sul ColorPicker;
- parent = nullptr : QWidget*
  Puntatore al QWidget padre del ColorPicker.

`+ void setColor(color : QColor, )`

Imposta il colore nel widget

Argomenti

- color : QColor
  Colore da impostare.

`+ QColor getColor()`

Restituisce il colore selezionato nel widget

Note

- Deve essere esplicitamente marcato come costante.

`# void mouseReleaseEvent(evento : QMouseEvent * , )`

Metodo eseguito quando viene rilasciato il tasto del mouse premuto

Argomenti

- evento : QMouseEvent *
  Oggetto contenente tutti i parametri sull'evento avvenuto.

`# void changeEvent(event : QEvent * , )`

Metodo eseguito quando viene cambiato il colore nel widget

Argomenti

- event : QEvent *
  Oggetto contenente tutti i parametri sull'evento avvenuto.

`# void colorChanged(color : QColor, )`

Segnale emesso quando viene cambiato il colore del widget

Argomenti

- color : QColor
  Il colore che è appena stato cambiato.

3.15 DDDMob::C3DObject

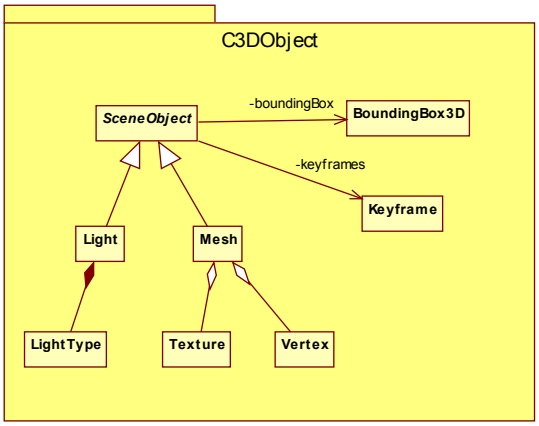

Figura 62: Componente DDDMob::C3DObject

Componente condiviso tra Modello e Controller che contiene la definizione dell'oggetto e mette a disposizione le operazioni di modifica

3.15.1 SceneObject (abstract)

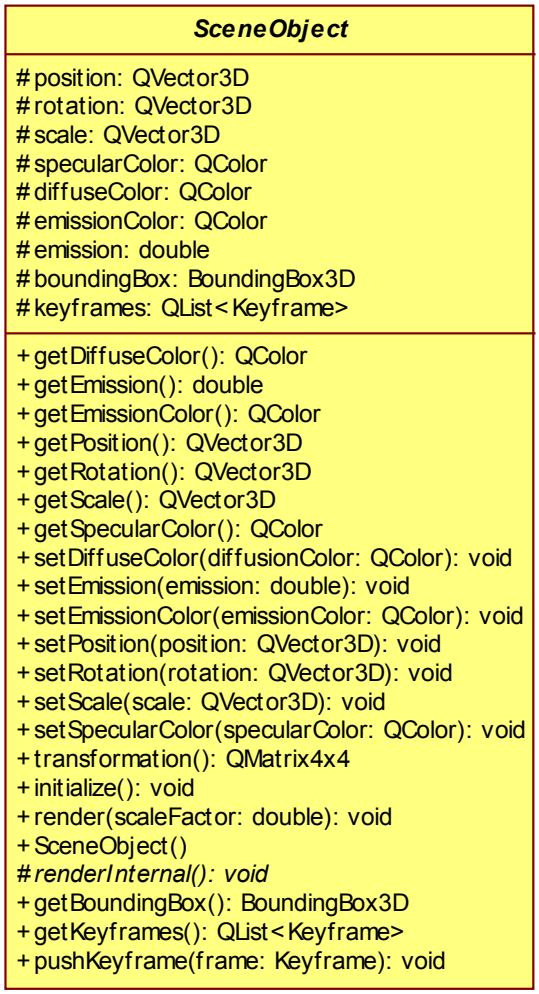

Figura 63: Classe SceneObject

Descrizione

Classe astratta che rappresenta gli oggetti che è possibile trovare nella scena$_G$ 3D;

Utilizzo

Mette a disposizione i metodi per la modifica degli attributi e per le trasformazioni all'oggetto;

Ereditata da

- DDDMob :: C3DObject :: Light;
- DDDMob :: C3DObject :: Mesh.

Attributi

`# QVector3D position`
Posizione;

`# QVector3D rotation`
Rotazione;

`# QVector3D scale`
Ridimensionamento;

`# QColor specularColor`
Colore speculare$_G$;

`# QColor diffuseColor`
Colore di diffusione$_G$;

`# QColor emissionColor`
Colore di emissione;

`# double emission`
Quantità di emissione nell'intervallo [0,1];

`# BoundingBox3D boundingBox`
Parallelepipedo di volume minimo che contiene completamente la parte visibile dell'oggetto;

`# QList<Keyframe> keyframes`
Sequenza di elementi di animazione.

Metodi

`+ QColor getDiffuseColor()`
Restituisce il colore di diffusione$_G$
Note
- Deve essere esplicitamente marcato come costante.

`+ double getEmission()`
Ritorna l'intensità di emissione
Note
- Deve essere esplicitamente marcato come costante.

`+ QColor getEmissionColor()`
Restituisce il colore emissione dell'oggetto
Note
- Deve essere esplicitamente marcato come costante.

`+ QVector3D getPosition()`
Ritorna la posizione dell'oggetto
Note
- Deve essere esplicitamente marcato come costante.

`+ QVector3D getRotation()`
Ritorna il parametro di rotazione dell'oggetto
Note
- Deve essere esplicitamente marcato come costante.

`+ QVector3D getScale()`
Ritorna la dimensione dell'oggetto
Note
- Deve essere esplicitamente marcato come costante.

`+ QColor getSpecularColor()`
Ritorna il colore speculare$_G$ dell'oggetto
Note

- Deve essere esplicitamente marcato come costante.

`+ void setDiffuseColor(diffusionColor : QColor, )`

Imposta il colore di diffusione$_G$ dell'oggetto

Argomenti

- diffusionColor : QColor
  Colore di diffusione$_G$ che si vuole impostare all'oggetto.

`+ void setEmission(emission : double, )`

Imposta il valore di emissione dell'oggetto

Argomenti

- emission : double
  Valore di emissione dell'oggetto che si vuole impostare.

`+ void setEmissionColor(emissionColor : QColor, )`

Imposta il colore di emissione dell'oggetto

Argomenti

- emissionColor : QColor
  Colore di emissione che si vuole impostare.

`+ void setPosition(position : QVector3D, )`

Imposta la posizione in X, Y e Z dell'oggetto

Argomenti

- position : QVector3D
  Coordinate della nuova posizione dell'oggetto.

`+ void setRotation(rotation : QVector3D, )`

Imposta la rotazione dell'oggetto

Argomenti

- rotation : QVector3D
  Rotazione nei tre assi che si vuole sia applicata all'oggetto.

`+ void setScale(scale : QVector3D, )`

Imposta la dimensione dell'oggetto

Argomenti

- scale : QVector3D
  Parametri di scala nei tre assi che si vuole siano applicati all'oggetto.

`+ void setSpecularColor(specularColor : QColor, )`

Imposta il colore speculare$_G$ dell'oggetto

Argomenti

- specularColor : QColor
  Colore che si vuole impostare come colore speculare$_G$ dell'oggetto.

`+ QMatrix4x4 transformation()`

Ritorna la matrice di trasformazione risultante da translation, rotation e scale

Note

- Deve essere esplicitamente marcato come costante.

`+ void render()`

Applica le trasformazioni ed esegue il rendering$_G$

Note

- Deve essere esplicitamente marcato come costante.

`+ SceneObject()`
Costruttore di default della classe SceneObject

`# void renderInternal()`
Esegue il rendering$_G$ in OpenGL$_G$
Note

- Deve essere astratto;
- Deve essere esplicitamente marcato come costante.

`+ BoundingBox3D getBoundingBox()`
Ritorna il parallelepipedo di volume minimo che contiene completamente la parte visibile dell'oggetto
Note

- Deve essere esplicitamente marcato come costante.

`+ QList<Keyframe> getKeyframes()`
Restituisce la lista dei Keyframe$_G$ dell'oggetto
Note

- Deve essere esplicitamente marcato come costante.

`+ void pushKeyframe(frame : Keyframe, )`
Aggiunge un keyframe$_G$
Argomenti

- frame : Keyframe$_G$
  Valore da inserire.

3.15.2 Light (class)

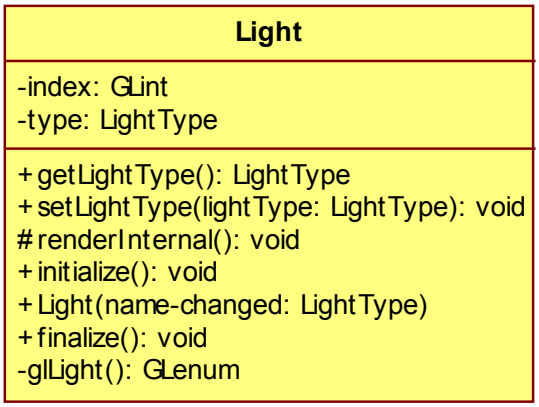

Figura 64: Classe Light

Descrizione
Classe che rappresenta una luce presente nella scena$_G$ 3D;

Utilizzo
Mette a disposizione i metodi per la modifica degli attributi propri della luce;

Classi ereditate

- DDDMob :: C3DObject :: SceneObject.

Attributi

`- GLint index`
Ritorna la luce OpenGL$_G$ dato l'indice;

`- LightType type`
Tipo della luce.

Metodi

`+ LightType getLightType()`
Ritorna il tipo di luce
Note
- Deve essere esplicitamente marcato come costante.

`+ void setLightType(lightType : LightType, )`
Imposta il tipo di luce
Argomenti
- lightType : LightType
  Il tipo di luce da impostare.

`# void renderInternal()`
Esegue il rendering$_G$ in OpenGL$_G$. Al programmatore è richiesta l'implementazione nell'ordine seguente:
- Rendering$_G$ del corretto simbolo segnaposto della luce mediante i metodi di utilità della classe *OpenGLUtils*: un cono od una sfera a seconda del tipo SPOT o OMNI della luce;
- Chiamata alle funzioni per definire le proprietà geometriche della luce (posizione, rotazione);
- Chiamata alle funzioni per definire le proprietà di colore di diffusione$_G$ e speculare della luce.

per un'implementazione corretta di quanto richiesto
Note
- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`+ void initialize()`
Inizializzazione della classe
Note
- Questo metodo è stato ridefinito.

`+ Light(type = LightType::OMNI : LightType , )`
Costruttore della classe
Argomenti
- type = LightType::OMNI : LightType
  Tipo di luce da impostare.

`+ void finalize()`
Finalizzazione della classe
Note

- Questo metodo è stato ridefinito.

`- GLenum glLight()`
Ritorna la luce OpenGL$_G$ dato l'indice
Note
- Deve essere esplicitamente marcato come costante.

3.15.3 Mesh (class)

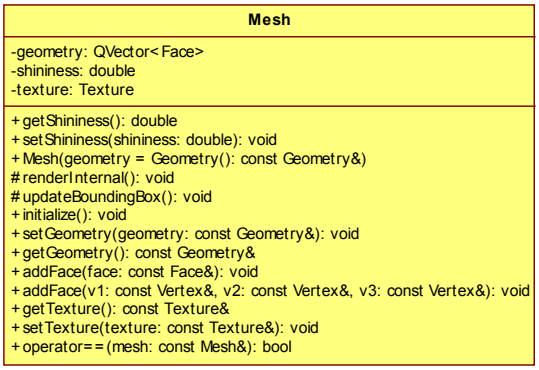

Figura 65: Classe Mesh

Descrizione
Classe che rappresenta una mesh poligonale presente nella scena$_G$ 3D;

Utilizzo
Mette a disposizione i metodi per la modifica degli attributi propri delle mesh;

Classi ereditate

- DDDMob :: C3DObject :: SceneObject.

Attributi

`- QVector<Face> geometry`
Vettore di facce;

`- double shininess`
Lucentezza;

`- Texture texture`
Texture.

Metodi

`+ double getShininess()`
Ritorna la lucentezza della Mesh
Note
- Deve essere esplicitamente marcato come costante.

`+ void setShininess(shininess : double, )`
Modifica la lucentezza
Argomenti

- shininess : double
  Lucentezza da impostare sull'oggetto.

`+ Mesh(geometry = Geometry() : const Geometry&, )`

Costruttore di Mesh

Argomenti

- geometry = Geometry() : const Geometry&
  Riferimento all'oggetto 3D della libreria Qt3D che si vuole rappresentare.

`# void renderInternal()`

Esegue il rendering$_G$ in OpenGL$_G$. Al programmatore è richiesta l'implementazione nell'ordine seguente:

- chiamata al rendering$_G$ della Texture;
- Chiamate per l'impostazione dei corretti valori di materiale dell'oggetto: colore di diffusione$_G$, speculare, lucentezza ed emissività;
- Rendering$_G$ di ogni triangolo grazie alla chiamata del metodo *render()* di ogni faccia della mesh.

per un'implementazione corretta di quanto richiesto

Note

- Deve essere virtuale;
- Deve essere esplicitamente marcato come costante;
- Questo metodo è stato ridefinito.

`# void updateBoundingBox()`

Aggiorna BoundingBox$_G$

`+ void initialize()`

Inizializzione della classe

Note

- Questo metodo è stato ridefinito.

`+ void setGeometry(geometry : const Geometry&, )`

Imposta la geometria dell'oggetto

Argomenti

- geometry : const Geometry&
  Geometria da impostare.

`+ const Geometry& getGeometry()`

Restituisce la geometria dell'oggetto

Note

- Deve essere esplicitamente marcato come costante.

`+ void addFace(face : const Face&, )`

Aggiunge una faccia all'oggetto

Argomenti

- face : const Face&
  Faccia da aggiungere all'oggetto.

`+ void addFace(v1 : const Vertex& , v2 : const Vertex& , v3 : const Vertex&, )`

Aggiunge una faccia all'oggetto

Argomenti

- v1 : const Vertex&
  Vertice 1 della faccia da aggiungere;
- v2 : const Vertex&
  Vertice 2 della faccia da aggiungere;
- v3 : const Vertex&
  Vertice 3 della faccia da aggiungere.

`+ const Texture& getTexture()`
Ritorna la texture dell'oggetto
Note

- Deve essere esplicitamente marcato come costante.

`+ void setTexture(texture : const Texture&, )`
Imposta la texture dell'oggetto
Argomenti

- texture : const Texture&
  Texture da impostare.

`+ bool operator== (mesh : const Mesh& , )`
Ridefinizione dell'operatore ==
Argomenti

- mesh : const Mesh&
  Riferimento all'oggetto Mesh.

Note

- Deve essere esplicitamente marcato come costante.

3.15.4 LightType (class)

Figura 66: Classe LightType

Descrizione
Definisce il tipo di luce;

Utilizzo
Viene utilizzato come `Enumerate`.
I valori possibili sono:

- OMNI;
- SPOT.

3.15.5 Texture (class)

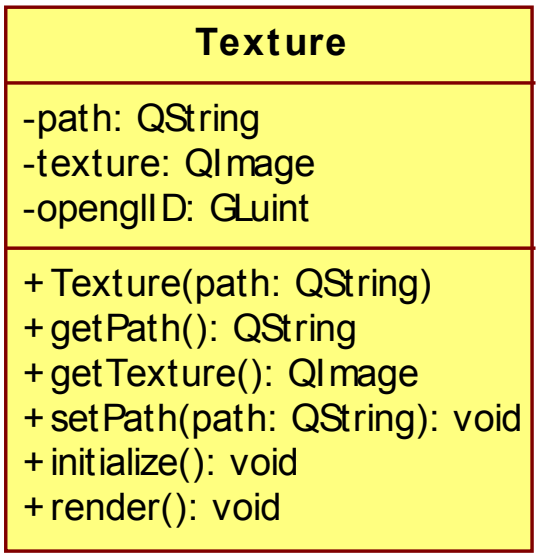

Figura 67: Classe Texture

Descrizione

Classe che rappresenta una texture per un oggetto della scena$_G$;

Utilizzo

Classe utilizzata per la rappresentazione di una texture per un oggetto della scena$_G$, definita con il percorso al file immagine e con una rappresentazione interna dell'immagine grazie alla classe QImage;

Attributi

`- QString path`
Path della Texture;

`- QImage texture`
Immagine della Texture;

`- GLuint openglID`
Id in openGL$_G$.

Metodi

`+ Texture(path : QString , )`
Costruttore per Texture, che si occupa di impostare il percorso fornito e caricare dinamicamente in una QImage l'immagine trovata

Argomenti

- path : QString
  Percorso del file dove risiede la texture.

`+ QString getPath()`
Ritorna il percorso del file immagine della texture

Note

- Deve essere esplicitamente marcato come costante.

`+ QImage getTexture()`

Ritorna il file immagine della texture caricato dinamicamente come QImage
Note

- Deve essere esplicitamente marcato come costante.

`+ void setPath(path : QString, )`

Imposta il percorso del file immagine della texture
Argomenti

- path : QString
  Percorso del file dove risiede la texture.

`+ void initialize()`

Inizializza l'oggetto per il rendering$_G$ di OpenGL$_G$. Al programmatore è richiesto:

- Chiamare i metodi di impostazione dei parametri della texture:
  - glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
  - glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  - glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  - glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  - glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR).
- Eseguire il rendering$_G$ della texture grazie allo stream$_G$ di byte dell'immagine.

per un'implementazione corretta di quanto richiesto

`+ void render()`

Esegue il render della texture. Al programmatore è richiesto:

1. Chiamare il metodo OpenGL$_G$ per il rendering$_G$ di una texture bidimensionale;
2. Bindare la texture al corretto id OpenGL$_G$ presente come campo dati della classe.

per un'implementazione corretta di quanto richiesto
Note

- Deve essere esplicitamente marcato come costante.

3.15.6 Vertex (class)

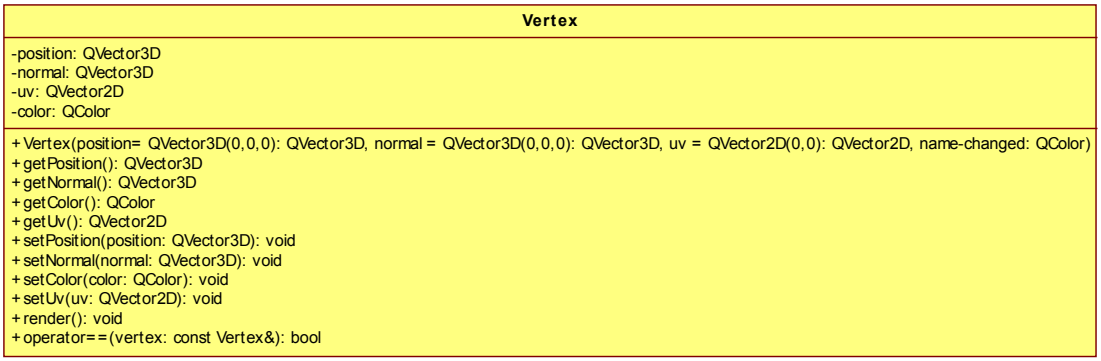

Figura 68: Classe Vertex

Descrizione

Classe che rappresenta un vertice di un oggetto;

Utilizzo

Viene usata per fornire informazioni utili proprie dei vertici, quali la posizione, le normali e l'UV;

Attributi

`- QVector3D position`
Posizione con x,y,z;

`- QVector3D normal`
Normale in x,y,z;

`- QVector2D uv`
Coordinate nello spazio della texture;

`- QColor color`
Colore dell'oggetto.

Metodi

`+ Vertex(position= QVector3D(0,0,0) : QVector3D, normal = QVector3D(0,0,0) : QVector3D, uv = QVector2D(0,0) : QVector2D, color = Qt::transparent : QColor, )`
Costruttore dell'oggetto Vertex
Argomenti

- position= QVector3D(0,0,0) : QVector3D
  Posizione del vertice;
- normal = QVector3D(0,0,0) : QVector3D
  Normale del vertice;
- uv = QVector2D(0,0) : QVector2D
  Coordinate UV del vertice;
- color = Qt_G::transparent : QColor
  Colore del vertice.

`+ QVector3D getPosition()`
Ritorna la posizione del vertice
Note
- Deve essere esplicitamente marcato come costante.

`+ QVector3D getNormal()`
Ritorna la normale del vertice
Note
- Deve essere esplicitamente marcato come costante.

`+ QColor getColor()`
Torna il colore del vertice
Note
- Deve essere esplicitamente marcato come costante.

`+ QVector2D getUv()`
Restituisce le coordinate nello spazio della texture
Note
- Deve essere esplicitamente marcato come costante.

`+ void setPosition(position : QVector3D , )`
Imposta la posizione del vertice
Argomenti
- position : QVector3D
  Posizione del vertice.

`+ void setNormal(normal : QVector3D , )`
Imposta i valori della normale
Argomenti
- normal : QVector3D
  Normale da impostare.

`+ void setColor(color : QColor, )`
Imposta il colore del vertice
Argomenti
- color : QColor
  Colore da impostare.

`+ void setUv(uv : QVector2D, )`
Imposta le coordinate nello spazio della texture
Argomenti
- uv : QVector2D
  Coordinate UV.

`+ void render()`
Esegue il render del vertice. Al programmatore è richiesto il richiamo della funzione per il rendering$_G$ di vettori tridimensionali presente in *OpenGLUtils*
Note
- Deve essere esplicitamente marcato come costante.

`+ bool operator== (vertex : const Vertex& , )`
Redefinizione dell'operatore ==
Argomenti

- vertex : const Vertex&
  Riferimento all'oggetto Vertex.

Note

- Deve essere esplicitamente marcato come costante.

3.15.7 Keyframe (class)

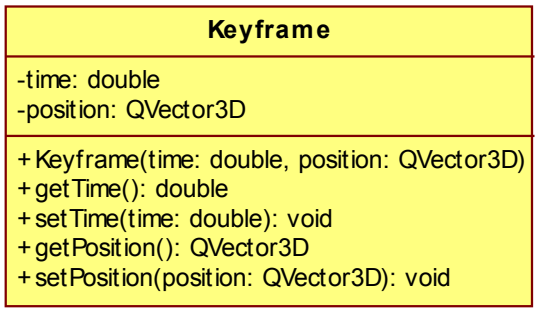

Figura 69: Classe Keyframe

Descrizione

Classe che rappresenta un keyframe$_G$ nella scena$_G$ 3D;

Utilizzo

Viene utilizzata per contenere le informazioni di un dato keyframe$_G$, tempo e posizione, per uno degli oggetti della scena$_G$;

Attributi

`- double time`
Durata del keyframe$_G$;

`- QVector3D position`
Posizione del keyframe$_G$.

Metodi

`+ Keyframe(time : double , position : QVector3D , )`
Costruttore dell'oggetto Keyframe$_G$
Argomenti

- time : double
  Durata del Keyframe$_G$;
- position : QVector3D
  Posizione del Keyframe$_G$.

`+ double getTime()`
Ritorna la durata del keyframe$_G$

`+ void setTime(time : double , )`
Imposta la durata del keyframe$_G$
Argomenti

- time : double
  Durata del Keyframe$_G$.

`+ QVector3D getPosition()`
Ritorna la posizione del keyframe$_G$

`+ void setPosition(position : QVector3D, )`
Imposta la posizione del keyframe$_G$
Argomenti

- position : QVector3D
  Posizione del Keyframe$_G$ da impostare.

3.15.8 BoundingBox3D (class)

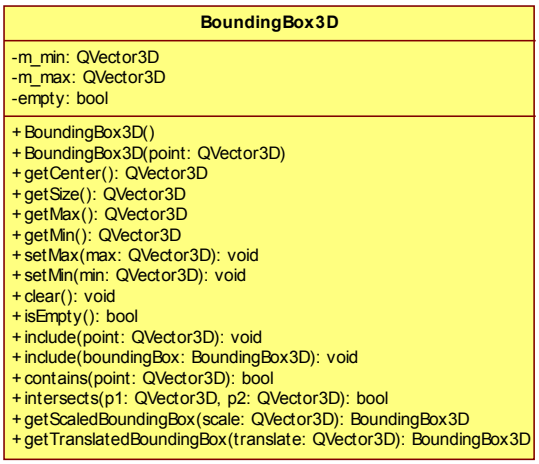

Figura 70: Classe BoundingBox3D

Descrizione
Oggetto Bounding Box$_G$;

Utilizzo

Attributi

`- QVector3D m_min`
Vertice minore;

`- QVector3D m_max`
Vertice massimo;

`- bool empty`
Boundig Box vuoto.

Metodi

`+ BoundingBox3D()`
Costruttore di default di BoundingBox3D

`+ BoundingBox3D(point : QVector3D , )`
Costruttore di BoundingBox3D
Argomenti

- point : QVector3D
  Punto massimo e minimo per creare la BoundingBox3D.

`+ QVector3D getCenter()`
Restituisce il centro del BoundingBox_G
Note

- Deve essere esplicitamente marcato come costante.

`+ QVector3D getSize()`
Restituisce altezza, larghezza, lunghezza della BoundingBox_G
Note

- Deve essere esplicitamente marcato come costante.

`+ QVector3D getMax()`
Ritorna il vertice massimo della BoundingBox_G
Note

- Deve essere esplicitamente marcato come costante.

`+ QVector3D getMin()`
Ritorna il vertice minimo della BoundingBox_G
Note

- Deve essere esplicitamente marcato come costante.

`+ void setMax(max : QVector3D , )`
Imposta il vertice massimo del BoundingBox_G
Argomenti

- max : QVector3D
  Vertice massimo da impostare.

`+ void setMin(min : QVector3D, )`
Imposta il vertice minimo del BoundingBox_G
Argomenti

- min : QVector3D
  Vertice minimo da impostare.

`+ void clear()`
Azzera il BoundingBox_G

`+ bool isEmpty()`
Ritorna true se il BoundingBox_G è vuoto
Note

- Deve essere esplicitamente marcato come costante.

`+ void include(point : QVector3D , )`
Espande il BoundingBox3D fino ad includere il punto (x,y,z)
Argomenti

- point : QVector3D
  Punto da includere nella BoundingBox3D.

`+ void include(boundingBox : BoundingBox3D , )`

Espande il BoundingBox3D fino a includere il BoundingBox_G passato per parametro

Argomenti

- boundingBox_G : BoundingBox3D
  BoundingBox3D da includere.

`+ bool contains(point : QVector3D , )`

Ritorna true se il BoundingBox3D contiene il punto (x,y,z)

Argomenti

- point : QVector3D
  Punto da controllare.

Note

- Deve essere esplicitamente marcato come costante.

`+ bool intersects(p1 : QVector3D , p2 : QVector3D, )`

Restituisce true se il BoundingBox3D interseca l'area definita dai due punti passati

Argomenti

- p1 : QVector3D
  Primo punto che identifica l'area da controllare;
- p2 : QVector3D
  Secondo punto che identifica l'area da controllare.

Note

- Deve essere esplicitamente marcato come costante.

`+ BoundingBox3D getScaledBoundingBox(scale : QVector3D , )`

Ritorna un BoundingBox_G scalato

Argomenti

- scale : QVector3D
  Vettore usato per scalare la BoundingBox3D.

Note

- Deve essere esplicitamente marcato come costante.

`+ BoundingBox3D getTranslatedBoundingBox(translate : QVector3D, )`

Ritorna un BoundingBox_G translato

Argomenti

- translate : QVector3D
  Vettore usato per translare la BoundingBox3D.

Note

- Deve essere esplicitamente marcato come costante.

4 Diagrammi di sequenza

Vengono qui riportati i diagrammi di sequenza delle operazioni principali dell'applicazione.

4.1 Importazione

Il diagramma in figura 71 rappresenta l'interazione che avviene tra i componenti dell'applicativo nel momento in cui l'utente desidera avviare l'apertura di un file.
La sequenza di azioni che portano all'apertura di un file viene scatenata dall'utente tramite l'invocazione del comando *Open* del menu *File*. A questo punto viene invocato il *Controller* il quale si occupa di creare e aprire la finestra di dialogo dalla quale l'utente può selezionare il file desiderato; in seguito viene recuperato il nome ed il percorso del file che sono necessari per effettuare l'importazione.
Di conseguenza viene invocata la classe *SceneAdapter* la quale si occupa di leggere il file tramite la libreria Assimp e di salvare nel *Model* i dati della scena$_G$. In seguito *SceneAdapter* lancia il segnale *signalUpdatedScene()* che viene intercettato da 3 oggetti: *GLView* la quale si occupa di renderizzare la scena$_G$ nella finestra sfruttando le libreria OpenGL$_G$; *Selector* il quale aggiorna il widget *Object Selector* che visualizza l'elenco degli oggetti della scena$_G$ e permette di selezionarli e infine *Transformation* che aggiorna l'omonimo widget impostando gli slider con i parametri della scena$_G$.
A questo punto il file è aperto e la scena$_G$ è visualizzata nell'interfaccia grafica e l'utente può proseguire con altre azioni.

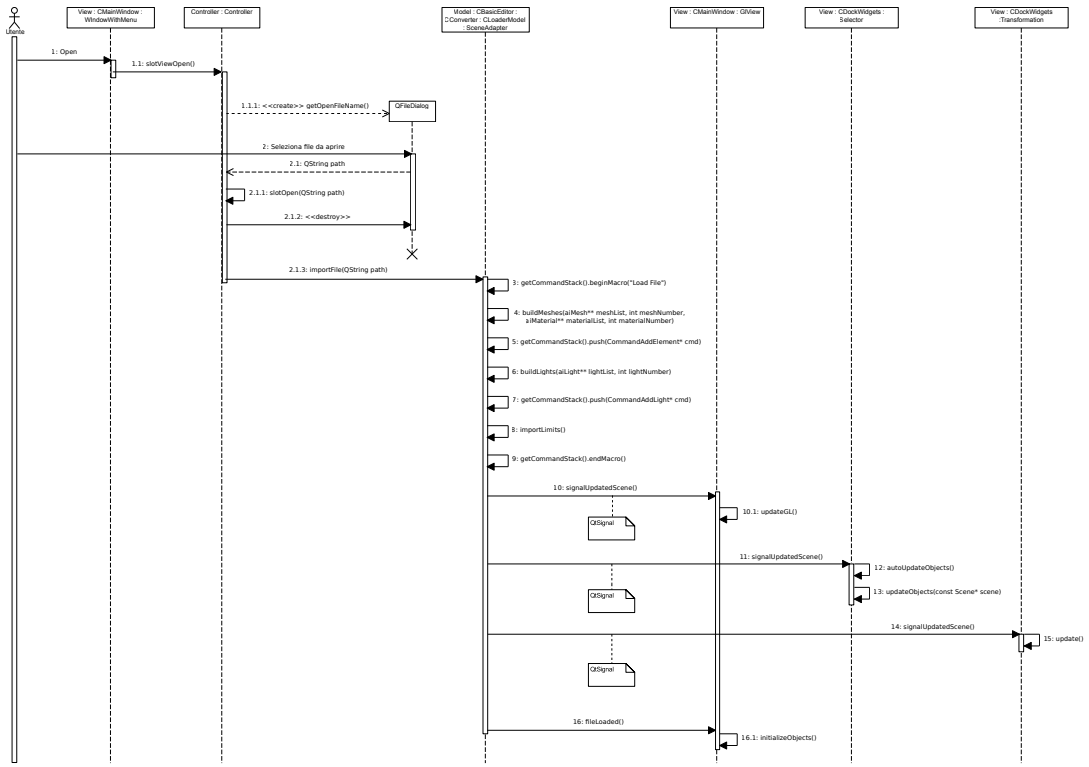

Figura 71: Diagramma di sequenza per l'importazione di un file

4.2 Esportazione

Il diagramma in figura 72 rappresenta l'interazione che avviene tra i componenti dell'applicativo nel momento in cui l'utente desidera avviare il salvataggio di un file.
La sequenza di azioni che portano al salvataggio di un file viene scatenata dall'utente tramite l'invocazione del comando *Save as* del menu *File*. A questo punto viene invocato il *Controller* il quale si occupa di creare e aprire la finestra di dialogo dalla quale l'utente può selezionare il formato del file da esportare e inserire il nome del file dopo averne indicato il percorso.
In seguito viene invocata la classe *SceneAdapter* che chiama *SceneExporter* passando il riferimento alla scena$_G$ corrente, il percorso di salvataggio del file e il tipo scelto dall'utente; a questo punto viene creato il corretto exporter, in questo caso *ExporterJSON*, che si preoccupa di serializzare la scena$_G$ contenente i diversi oggetti e le luci e salvarli correttamente sul file prescelto.

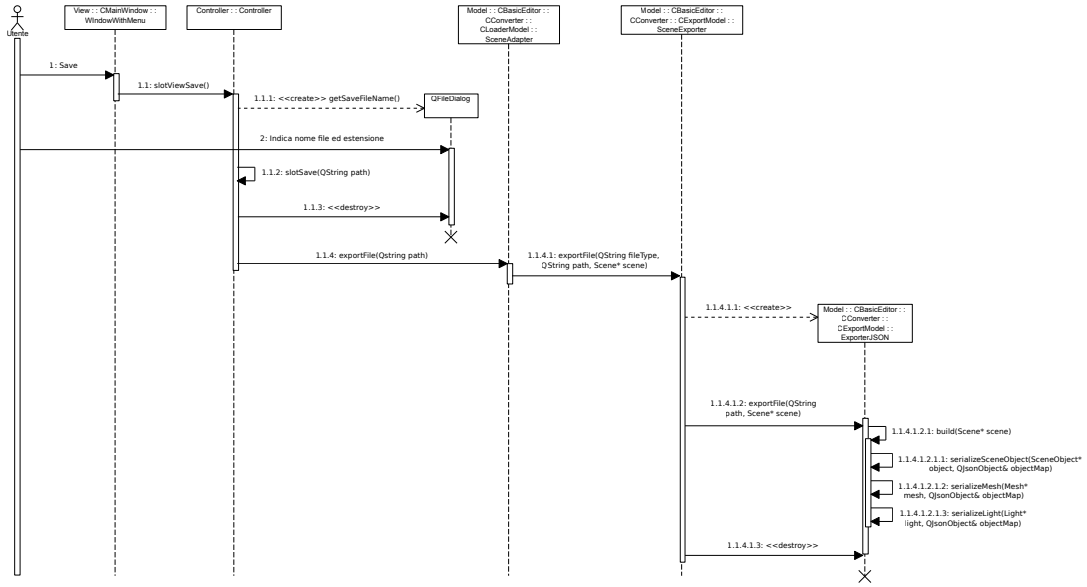

Figura 72: Diagramma di sequenza per l'esportazione di un file

4.3 Traslazione

Il diagramma in figura 73 rappresenta l'interazione che avviene tra i componenti dell'applicativo nel momento in cui l'utente desidera effettuare una traslazione ad un oggetto presente nella scena$_G$.
Per poter effettuare una traslazione l'utente deve per prima cosa selezionare l'oggetto desiderato; non appena l'utente esegue la selezione mediante la pressione del tasto destro del mouse, la classe *GLView* rileva l'evento e invoca la classe *SceneAdapter* avvisandola dell'avvenuta selezione; a quel punto viene invocato il *Selector* il quale evidenzia nella lista degli oggetti della scena$_G$ l'oggetto che è stato selezionato.
Nel momento in cui l'utente rilascia il pulsante destro la *GLView* rileva l'evento e disegna intorno all'oggetto selezionato un box per evidenziare l'avvenuta selezione e aggiornando di conseguenza la scena$_G$.
A questo punto l'utente è libero di traslare l'oggetto in qualunque posizione: per fare questo deve premere il pulsante sinistro e trascinare l'oggetto nella posizione desiderata. L'evento scatenato dal trascinamento del mouse viene catturato dalla *GLView* la

quale recupera l'oggetto selezionato da *SsceneAdapter* e dello stesso oggetto recupera la posizione attuale. Durante il trascinamento viene calcolato lo spostamento come delta tra posizione iniziale e l'ultima rilevata.
Tale parametro delta viene passato al *Controller*, dopo averne recuperato il riferimento, il quale si occupa di recuperare da *SceneAdapter* l'oggetto corrente e il *commandStack* attuale e di creare il *commandPosition* il quale applicherà la trasformazione al solido traslato.
Come ultima azione *GLView* rileva il rilascio del pulsante del mouse e aggiorna l'anteprima della scena$_G$.

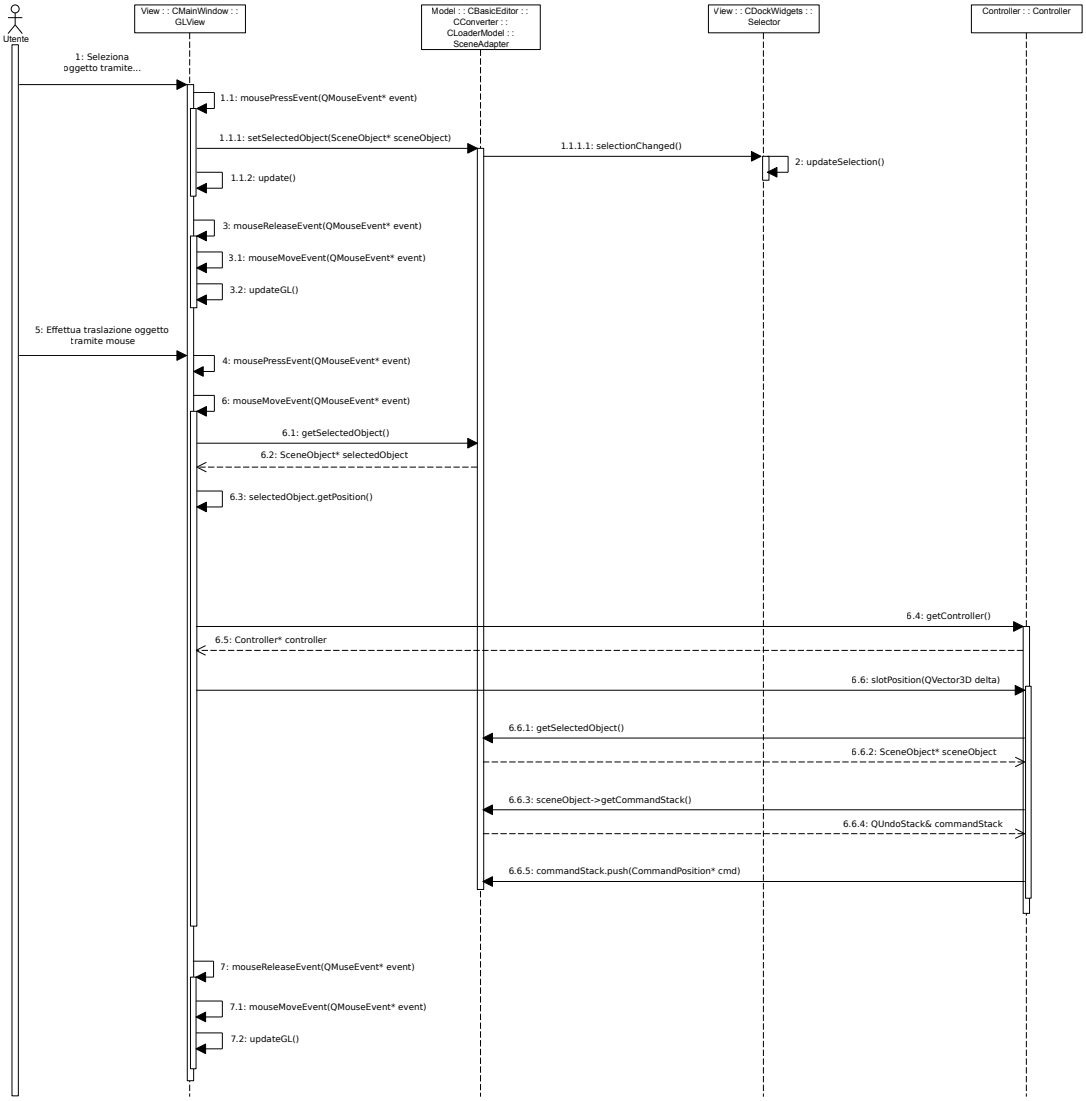

Figura 73: Diagramma di sequenza per la traslazione di un oggetto

A Tracciamento

A.1 Tracciamento requisiti - classi

A.2 Tracciamento classi - requisiti

Nella tabella sottostante sono presenti delle celle vuote in corrispondenza di alcune classi concrete, a causa del tracciamento dei requisiti all'interfaccia che espone i loro metodi.

Classe	Requisiti
Scene	
SceneAdapter	R0F8.1 R1F8.4 R1F8.4.1 R2F8.3.2
SceneExporter	R0F1 R0F1.3 R0F1.3.1 R0F1.3.2 R0F1.3.3 R0F1.3.4 R1F1.3.5
ExporterUBJSON	R2F1.7
ExporterXML	R0F1.1 R2F1.5 R2F1.5.1
ExporterMinifiedJSON	R0F1.1
Controller	
CommandTransform	
CommandAddLight	R1F8.3.4 R2F8.3.4.1
SceneObject	R1F8.3 R2F8.3.1.1
Light	
Mesh	
Setting	R1F1.6.3 R1F7.1.1.1 R2F1.6 R2F1.6.1 R2F1.6.2
AxisSlider	
ColorPicker	
LightType	
Material	R1F8.3.1.1.4 R1F8.3.1.1.4.1 R1F8.3.1.1.4.2 R1F8.3.1.1.5 R1F8.3.1.2.1 R1F8.3.1.2.2
Selector	R1F8.3.5

LightWidget	R1F8.3.1.2.3 R1F8.3.4 R2F8.3.1.2 R2F8.3.4.1
Action	R2F8.3.2
Transformation	R0F8.3.1.2.4 R2F8.3.1 R2F8.3.1.2.5
WindowWithMenu	
WindowWithDock	
WindowWithStatus	R1F15
Decorator	
MainWindow	R2F7
View3D	
DeviceLimit	R2F1.6
CommandEditMesh	
CommandEditLight	R1F8.3.1.2.3 R2F8.3.1.2
SettingsWindow	
CommandScene	R1F8.3 R2F8.3.2
CommandAddElement	
FacadeView	R0F8 R2F7 R2F7.3 R2F7.3.1 R2F7.3.2
ErrorMessageBox	R2F7.2 R2F7.2.1 R2F7.2.2 R2F7.2.3
About	R2F7.4
NumericPrecisionType	R0F1.2
CommandPosition	R1F8.3.1.1.2
CommandScale	R1F8.3.1.1.3
CommandRotation	R1F8.3.1.1.1
CommandLightType	
CommandShininess	
CommandEmission	R1F8.3.1.1.4 R1F8.3.1.2.1
CommandColor	R1F8.3.1.2.1
CommandColorEmission	
CommandColorSpecular	R1F8.3.1.1.4.1
CommandColorDiffuse	R1F8.3.1.1.4.2
CommandEditObject	R2F8.3.1.1

GLView	R0F8.3.3 R1F8.3 R1F8.3.1.1.1 R1F8.3.1.1.2 R1F8.3.1.1.3 R1F8.3.5 R2F7.1 R2F7.1.1 R2F7.1.2 R2F7.1.2.1 R2F7.1.2.2 R2F7.1.2.3
ImporterJson	R2F8.2
Texture	R0F1.3.2
Vertex	R0F1.3.1
Keyframe_G	R1F1.3.5
BoundingBox3D	R1F8.3.5
ExporterJSON	R0F1.1 R0F1.4 R0F1.4.1
QGLWidget	

Tabella 2: Tabella classi / requisiti

A.3 Tracciamento modulo - test

Metodo	Test
CommandTransform::CommandTransform()	TU30
CommandAddLight::CommandAddLight()	TU19
CommandAddLight::undo()	
CommandAddLight::redo()	
CommandEditMesh::CommandEditMesh()	TU30
CommandEditMesh::getMesh()	TU31
CommandEditLight::CommandEditLight()	TU30
CommandEditLight::getLight()	TU31
CommandScene::CommandScene()	TU30
CommandAddElement::CommandAddElement()	TU19
CommandAddElement::undo()	TU19
CommandAddElement::redo()	TU19
CommandPosition::CommandPosition()	TU25
CommandPosition::id()	TU30
CommandPosition::redo()	TU25
CommandPosition::undo()	TU25
CommandPosition::mergeWith()	TU25
CommandScale::CommandScale()	TU27
CommandScale::redo()	TU27
CommandScale::undo()	TU27
CommandScale::id()	TU30
CommandScale::mergeWith()	TU27
CommandRotation::CommandRotation()	TU26
CommandRotation::id()	TU30
CommandRotation::redo()	TU26
CommandRotation::undo()	TU26
CommandRotation::mergeWith()	TU26
CommandLightType::CommandLightType()	TU24
CommandLightType::undo()	TU24
CommandLightType::redo()	TU24
CommandShininess::CommandShininess()	TU28
CommandShininess::undo()	TU28
CommandShininess::redo()	TU28
CommandShininess::id()	TU30
CommandShininess::mergeWith()	TU28
CommandEmission::CommandEmission()	TU23
CommandEmission::undo()	TU23
CommandEmission::redo()	TU23
CommandEmission::id()	TU30
CommandEmission::mergeWith()	TU23
CommandColor::CommandColor()	TU30
CommandColorEmission::CommandColorEmission()	TU21
CommandColorEmission::undo()	TU21
CommandColorEmission::redo()	TU21

CommandColorSpecular::CommandColorSpecular()	TU22
CommandColorSpecular::redo()	TU22
CommandColorSpecular::undo()	TU22
CommandColorDiffuse::CommandColorDiffuse()	TU20
CommandColorDiffuse::redo()	TU20
CommandColorDiffuse::undo()	TU20
CommandEditObject::CommandEditObject()	TU30
Setting::getSetting()	TU16
Setting::signalSettingsUpdate()	TU30
Setting::Setting()	TU16
Setting::setNumericPrecision()	TU32
Setting::setSceneBackgroundColor()	TU32
Setting::getNumericPrecision()	TU32
Setting::getSceneBackgroundColor()	TU32
Setting::getSelectedDevice()	TU32
Setting::selectDevice()	TU15
Setting::loadSettings()	
Setting::saveSettings()	
Setting::signalError()	
Setting::signalBackgroundColorUpdate()	
Setting::signalNumericPrecisionUpdate()	
Setting::getDeviceLimits()	
DeviceLimit::DeviceLimit()	TU15
SceneExporter::exportFile()	
SceneExporter::exportFile()	TU14
ExporterUBJSON::exportFile()	TU35
ExporterUBJSON::build()	TU35
ExporterUBJSON::serializeSceneObject()	TU35
ExporterUBJSON::serializeMesh()	TU35
ExporterUBJSON::serializeLight()	TU35
ExporterUBJSON::serialize()	TU35
ExporterUBJSON::serialize()	TU35
ExporterUBJSON::serialize()	TU35
ExporterXML::exportFile()	TU33
ExporterXML::build()	TU33
ExporterXML::streamSceneObject()	TU33
ExporterXML::streamMesh()	TU33
ExporterXML::streamLight()	TU33
ExporterXML::stream$_G$ ()	TU33
ExporterXML::stream$_G$ ()	TU33
ExporterMinifiedJSON::exportFile()	TU30
ExporterJSON::exportFile()	TU34
ExporterJSON::build()	TU34
ExporterJSON::serializeSceneObject()	TU34
ExporterJSON::serializeMesh()	TU34
ExporterJSON::serializeLight()	TU34
ExporterJSON::serialize()	TU34

ExporterJSON::serialize()	TU34
ExporterJSON::serialize()	TU34
Scene::getObject()	TU32
Scene::exportFile()	
Scene::importFile()	
Scene::signalUpdatedScene()	TU30
Scene::signalError()	TU30
Scene::getCommandStack()	TU32
Scene::getObjectNames()	TU32
Scene::getSelectedObject()	TU31
Scene::setSelectObject()	TU31
Scene::getCommandStack()	TU31
Scene::selectByName()	
Scene::getObjectType()	TU31
Scene::update()	TU30
Scene::selectionChanged()	
Scene::fileLoaded()	
Scene::getAllObjects()	TU31
Scene::addObject()	
Scene::getLightNumber()	
Scene::removeObject()	
SceneAdapter::SceneAdapter()	
SceneAdapter::getObject()	TU31
SceneAdapter::exportFile()	TU2
SceneAdapter::importFile()	TU1
SceneAdapter::signalUpdatedScene()	
SceneAdapter::signalError()	
SceneAdapter::getObjectNames()	
SceneAdapter::selectByName()	TU29
SceneAdapter::buildLights()	TU1
SceneAdapter::buildMeshes()	TU1
SceneAdapter::meshCreator()	TU1
SceneAdapter::lightCreator()	TU1
SceneAdapter::importLimits()	TU1
SceneAdapter::getLightNumber()	TU1
SceneAdapter::removeObject()	TU3
SceneAdapter::addObject()	TU1
SceneAdapter::getAllObjects()	TU1
ImporterJson::importFile ()	TU36
ImporterJson::readObject()	TU36
ImporterJson::readLight()	TU36
ImporterJson::readMesh()	TU36
ImporterJson::readObjectProperties()	TU36
Controller::slotRotation()	TU30
Controller::slotScale()	TU30
Controller::slotPosition()	TU30
Controller::slotAddLight()	TU30

Controller::slotColorDiffuse()	TU30
Controller::slotColorSpecular()	TU30
Controller::slotColorEmission()	TU30
Controller::slotSelectObject()	TU30
Controller::slotLightType()	TU30
Controller::slotMeshShininess()	TU30
Controller::slotEmission()	TU30
Controller::slotSettings()	TU30
Controller::slotOpen()	TU30
Controller::slotSave()	TU30
Controller::slotViewAbout()	TU30
Controller::slotViewSave()	TU30
Controller::slotViewOpen()	TU30
Controller::slotViewHelp()	TU30
Controller::slotViewSettings()	TU30
Controller::Controller()	TU30
Controller::getController()	TU32
Controller::getScene()	TU31
Controller::start3DMob()	TU30
Controller::getDataDir()	
FacadeView::getMainWindow()	TU32
FacadeView::showHelpWindow()	
FacadeView::showSettings()	
FacadeView::showAbout()	
FacadeView::showMainWindow()	
FacadeView::FacadeView()	
FacadeView::operator=()	
FacadeView::FacadeView()	
FacadeView::operator=()	
FacadeView::getErrorMessageBox()	
About::About()	
Material::setDiffuseColor()	TU32
Material::getDiffuseColor()	TU32
Material::getSpecularColor()	TU32
Material::getEmissionColor()	TU32
Material::setEmissionColor()	TU31
Material::getShininess()	TU31
Material::setShininess()	TU31
Material::Material()	
Material::setSpecularColor ()	TU31
Material::update()	
Selector::Selector()	
Selector::updateObjects()	
Selector::autoUpdateObjects()	
Selector::updateSelection()	
LightWidget::LightWidget()	TU30
LightWidget::getLightType()	TU31

LightWidget::update()	
LightWidget::lightTypeChanged()	
Action::Action()	
Action::setUndoStack()	TU31
Transformation::getPosition()	TU31
Transformation::getRotation()	TU31
Transformation::getScale()	TU31
Transformation::setPosition()	TU31
Transformation::setRotation()	TU31
Transformation::setScale()	TU31
Transformation::update()	
Transformation::scaleChanged()	
Transformation::positionChanged()	
Transformation::rotationChanged()	
WindowWithMenu::WindowWithMenu()	TU30
WindowWithMenu::draw()	
WindowWithMenu::getWrappedWindow()	TU31
WindowWithDock::WindowWithDock()	TU30
WindowWithDock::draw()	TU30
WindowWithDock::getWrappedWindow()	TU31
WindowWithStatus::WindowWithStatus()	TU30
WindowWithStatus::draw()	
WindowWithStatus::getWrappedWindow()	TU31
Decorator::Decorator()	
Decorator::getWrappedWindow()	TU31
MainWindow::draw()	
MainWindow::getWrappedWindow()	TU31
View3D::View3D()	TU30
View3D::getSceneModel()	TU31
View3D::draw()	
View3D::getWrappedWindow()	TU31
View3D::updateSelection()	
GLView::paintGL()	
GLView::GLView()	
GLView::getSceneModel()	TU31
GLView::initializeObjects()	
GLView::updateBackgroundColor()	
GLView::initializeGL()	
GLView::resizeGL()	
GLView::mousePressEvent()	
GLView::mouseMoveEvent()	
GLView::mouseReleaseEvent()	
GLView::wheelEvent()	
GLView::keyPressEvent()	
GLView::keyReleaseEvent()	
GLView::renderAxes()	
GLView::selectionChanged()	

SettingsWindow::SettingsWindow()	
SettingsWindow::getDouble()	TU32
SettingsWindow::getColor()	TU32
SettingsWindow::selectLimits()	
ErrorMessageBox::showError()	TU30
AxisSlider::AxisSlider()	
AxisSlider::setValue()	TU32
AxisSlider::getValue()	TU32
AxisSlider::axisSlid()	
ColorPicker::updateColor()	
ColorPicker::updateColorFromDialog()	
ColorPicker::ColorPicker()	
ColorPicker::ColorPicker()	
ColorPicker::setColor()	TU32
ColorPicker::getColor()	TU32
ColorPicker::mouseReleaseEvent()	
ColorPicker::changeEvent()	
ColorPicker::colorChanged()	
SceneObject::getDiffuseColor()	TU32
SceneObject::getEmission()	TU32
SceneObject::getEmissionColor()	TU32
SceneObject::getPosition()	TU32
SceneObject::getRotation()	TU32
SceneObject::getScale()	TU32
SceneObject::getSpecularColor()	TU32
SceneObject::setDiffuseColor()	TU8
SceneObject::setEmission()	TU7
SceneObject::setEmissionColor()	TU17
SceneObject::setPosition()	TU6
SceneObject::setRotation()	TU4
SceneObject::setScale()	TU5
SceneObject::setSpecularColor()	TU18
SceneObject::transformation()	TU4
SceneObject::render()	
SceneObject::SceneObject()	
SceneObject::renderInternal()	
SceneObject::getBoundingBox()	TU32
SceneObject::getKeyframes()	TU32
SceneObject::pushKeyframe()	
Light::getLightType()	TU32
Light::setLightType()	TU11
Light::renderInternal()	
Light::initialize()	
Light::Light()	
Light::finalize()	
Light::glLight()	
Mesh::getShininess()	TU32

Mesh::setShininess()	TU12
Mesh::Mesh()	
Mesh::renderInternal()	
Mesh::updateBoundingBox()	
Mesh::initialize()	
Mesh::setGeometry()	TU31
Mesh::getGeometry()	TU31
Mesh::addFace()	
Mesh::addFace()	
Mesh::getTexture()	TU31
Mesh::setTexture()	TU31
Mesh::operator== ()	
Texture::Texture()	
Texture::getPath()	TU31
Texture:: getTexture()	TU31
Texture::setPath()	TU31
Texture::initialize()	
Texture::render()	
Vertex::Vertex()	
Vertex::getPosition()	TU31
Vertex::getNormal()	TU31
Vertex::getColor()	TU31
Vertex::getUv()	TU31
Vertex::setPosition()	TU31
Vertex::setNormal()	TU31
Vertex::setColor()	TU31
Vertex::setUv()	TU31
Vertex::render()	
Vertex::operator== ()	
$Keyframe_G$::$Keyframe_G$ ()	
$Keyframe_G$::getTime()	TU31
$Keyframe_G$::setTime()	TU31
$Keyframe_G$::getPosition()	TU31
$Keyframe_G$::setPosition()	TU31
BoundingBox3D::BoundingBox3D()	
BoundingBox3D::BoundingBox3D()	
BoundingBox3D::getCenter()	
BoundingBox3D::getSize()	
BoundingBox3D::getMax()	
BoundingBox3D::getMin()	
BoundingBox3D::setMax()	TU31
BoundingBox3D::setMin()	TU32
BoundingBox3D::clear()	
BoundingBox3D::isEmpty()	
BoundingBox3D::include()	
BoundingBox3D::include()	
BoundingBox3D::contains()	

BoundingBox3D::intersects()	
BoundingBox3D::getScaledBoundingBox()	
BoundingBox3D::getTranslatedBoundingBox()	

Tabella 3: Tabella metodi / test unità

B Schemi file esportati

B.1 JSONSchema

Viene riportato il JSONSchema che i file $JSON_G$ esportati rispettano.
Tale schema permette di definire la struttura del file $JSON_G$ esportati e di validarli.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "3DMob",
  "description": "Schema del file JSON esportato da 3DMob",
  "type": "object",
  "items": {
    "title": "3DMob",
    "type": "object",
    "properties" : {
      ".*": {
        "description": "Mesh o Luce della scena",
        "type": "array",
        "items":{
          "type": "object",
          "properties" : {
            "object": {
              "description": "Oggetto 3D con caratteristiche condivise tra mesh
                  e luce",
              "type": "object",
              "properties": {
                "position": {
                  "type": "array",
                  "items": {
                    "type": "number"
                  },
                  "minItems": 3,
                  "maxItems": 3
                },
                "rotation": {
                  "type": "array",
                  "items": {
                    "type": "number"
                  },
                  "minItems": 3,
                  "maxItems": 3
                },
                "scale": {
                  "type": "array",
                  "items": {
                    "type": "number"
                  },
                  "minItems": 3,
                  "maxItems": 3
                },
                "diffuseColor":{
                  "type": "string"
                },
                "emission":{
                  "type": "number"
```

```
      },
      "emissionColor":{
        "type": "string"
      },
      "specularColor":{
        "type": "string"
      },
      "keyframes":{
        "description": "animazioni presenti sull'oggetto 3D",
        "type": "array",
        "items": {
          "title" : "animazione presente sulla scena",
          "type" : "object",
          "properties" : {
            "position": {
              "type": "array",
              "items": {
                "type": "number"
              },
              "minItems": 3,
              "maxItems": 3
            },
            "time":{
              "type": "number"
            }
          },
          "required": ["position", "time"]
        }
      }
    },
    "required": ["position", "rotation", "scale", "diffuseColor",
                 "emission", "emissionColor", "specularColor",
                     "keyframes"]
  },
  "mesh":{
    "title" : "caratteristiche uniche della mesh",
    "type" : "object",
    "properties" : {
      "geometry":{
        "type": "array",
        "items": {
          "title" : "facce della mesh",
          "type" : "array",
          "items":{
            "type": "object",
            "properties":{
              "position": {
                "type": "array",
                "items": {
                  "type": "number"
                },
                "minItems": 3,
                "maxItems": 3
              },
              "normal": {
                "type": "array",
```

```
                    "items": {
                      "type": "number"
                    },
                    "minItems": 3,
                    "maxItems": 3
                  },
                  "uv": {
                    "type": "array",
                    "items": {
                      "type": "number"
                    },
                    "minItems": 3,
                    "maxItems": 3
                  }
                },
                "required": ["position","normal","uv"]
              }
            }
          },
          "shininess":{
            "type": "number"
          },
          "texture":{
            "type": "string"
          }
        },
        "minItems": 0,
        "required": ["geometry","shininess","texture"]
      },
      "light":{
        "title" : "caratteristiche uniche della luce",
        "type" : "object",
        "properties" : {
          "lightType":{
            "type": "integer"
          }
        },
        "minItems": 0,
        "required": ["lightType"]
      }
    },
    "required": ["object"]
  }
 }
},
"required": [".*"]
}
}
```

B.2 XMLSchema

Viene riportato il XMLSchema che i file XML_G esportati rispettano.
Tale schema permette di definire la struttura del file XML_G esportati e di validarli.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="3dMob"
xmlns="3dMob"
elementFormDefault="qualified">

<xs:element name="scene" type ="Tscena" />

	<xs:complexType name="Tscena">
		<xs:sequence>
			<xs:element name="mesh" type="Tmesh" minOccurs="0"
			    maxOccurs="unbounded"/>
			<xs:element name="light" type="Tlight" minOccurs="0"
			    maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>

	<xs:complexType name="Tlight">
		<xs:sequence>
			<xs:element name="object" type="Tobject"/>
			<xs:element name="lightType" type="xs:integer"/>
			<xs:element name="keyframes" type="Tkeyframes"/>
		</xs:sequence>
		<xs:attribute name="name" type="xs:string"/>
	</xs:complexType>

	<xs:complexType name="Tmesh">
		<xs:sequence>
			<xs:element name="object" type="Tobject"/>
			<xs:element name="shininess" type="xs:integer"/>
			<xs:element name="alpha" type="xs:integer"/>
			<xs:element name="texture" type="xs:string"/>
			<xs:element name="geometry" type="Tgeometry"/>
			<xs:element name="keyframes" type="Tkeyframes"/>
		</xs:sequence>
		<xs:attribute name="name" type="xs:string"/>
	</xs:complexType>

	<xs:complexType name="Tobject">
		<xs:sequence>
			<xs:element name="diffuseColor" type="xs:string"/>
			<xs:element name="emission" type="xs:integer"/>
			<xs:element name="emissionColor" type="xs:string"/>
			<xs:element name="position" type="Tvector3D"/>
			<xs:element name="rotation" type="Tvector3D"/>
			<xs:element name="scale" type="Tvector3D"/>
			<xs:element name="specularColor" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>

	<xs:complexType name="Tvector3D">
		<xs:sequence>
			<xs:element name="x" type="xs:double"/>
			<xs:element name="y" type="xs:double"/>
			<xs:element name="z" type="xs:double"/>
		</xs:sequence>
	</xs:complexType>
```

```
<xs:complexType name="Tgeometry">
<xs:sequence>
	<xs:element name="face" type="Tface" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Tface">
	<xs:sequence>
		<xs:element name="vertex" type="Tvertex"
		    maxOccurs="unbounded"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="Tvertex">
	<xs:sequence>
		<xs:element name="position" type="Tvector3D"/>
		<xs:element name="normal" type="Tnormal"/>
		<xs:element name="uv" type="Tuv"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="Tnormal">
	<xs:sequence>
		<xs:element name="x" type="xs:double"/>
		<xs:element name="y" type="xs:double"/>
		<xs:element name="z" type="xs:double"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="Tuv">
	<xs:sequence>
		<xs:element name="x" type="xs:double"/>
		<xs:element name="y" type="xs:double"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="Tkeyframes">
	<xs:sequence>
		<xs:element name="time" type="Tkeyframe" minOccurs="0"
		    maxOccurs="unbounded"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="Tkeyframe">
	<xs:sequence>
		<xs:element name="time" type="xs:double"/>
		<xs:element name="position" type="Tvector3D"/>
	</xs:sequence>
</xs:complexType>

</xs:schema>
```

C Utilizzo del Qt Undo Framework

Per implementare il sistema di comandi presente nell'applicazione è stato utilizzato il Qt$_G$'s Undo Framework$_G$. Il framework$_G$ scelto segue il Design Pattern$_G$ Command e ci consente di implementare in maniera semplice la creazione, l'invio e l'annullamento dei comandi.

Per creare un nuovo comando è sufficiente:

- Creare una classe derivata da QUndoCommand;
- Ridefinire il metodo `redo` in modo che esegua l'azione prevista dal comando;
- Ridefinire il metodo `undo` in modo che annulli l'azione prevista dal comando;
- Se il comando può essere unito ad altri comandi:
  - Ridefinire il metodo `id` in modo che ritorni un intero che identifichi il comando;
  - Ridefinire il metodo `mergeWith` in modo che aggiunga al comando su cui viene invocato un altro comando dello stesso tipo.

Nel caso sia necessario unire due comandi, verrà invocato il metodo `mergeWith` sul primo comando, passando come parametro il secondo comando. Se i comandi hanno lo stesso id l'azione del comando passato verrà aggiunta al primo comando non hanno id coincidente l'azione verrà annullata e `mergeWith` ritornerà `false`.

Segue una lista dei comandi utilizzati nell'applicazione e gli id dei comandi che possono essere uniti.

id	Comando
0x0001	CommandPosition
0x0002	CommandRotation
0x0003	CommandScale
0x0010	CommandEmission
0x0020	CommandShininess
	CommandAddElement
	CommandAddLight
	CommandColor
	CommandColorDiffuse
	CommandColorEmission
	CommandColorSpecular
	CommandEditLight
	CommandEditMesh
	CommandLightType
	CommandScene
	CommandTransform

Tabella 4: Comandi e relativi id