LABORATORIO DI MULTIMEDIA II

Progetto "Asteroid Cargo"

// Di Donato Valerio
// Ferronato Giacomo

Application

Creazione della scena
Nel metodo Application::setupScene() viene creato ogni elemento della scena principale dell'applicazione:

Camera:
Viene creata una camera e ne viene modificato l'aspect ratio in modo che sia corretto per la viewport.

Overlay:
Vengono caricati gli overlay che saranno usati durante la partita

Luci:
Vengono create e settate le due luci presenti nella scena: una ambient e una directional

Casse:
Attraverso una struttura "for" vengono create nella scena le casse: per fare ciò a ogni ciclo viene caricata e associata a un nodo di scena una mesh. A questa entità sarà poi associato un query flag, associato agli oggetti principali della missione, che servirà nel meccanismo di attracco di una cassa attraverso l'uso di una SphereQuery. Questi scene node saranno posizionati nello spazio attraverso una funzione randomica che li disporrà sugli assi x, y e z in una posizione compresa tra -5000 e 5000.

Malus:
Con la stessa tecnica delle casse saranno poi creati i malus, e ad ognuno sarà assegnato un query flag relativo al tipo di effetto negativo che dovrà essere rilevato dallo SphereQuery.

Il Portale:
La creazione del portale è suddivisa in tre fasi:

1. Nella prima fase vengono create le sfere che rappresenteranno il numero di casse da dover riportare al portale per completare la partita
2. Nella seconda si inseriscono nella scena le mesh che compongono la struttura del portale, e le si associa ai loro relativi nodi di scena

La disposizione spaziale a circonferenza delle sfere è data dall'uso di funzioni di seno e coseno per le due coordinate su cui si vuole disporre le sfere.
Durante la prima fase viene anche deciso in modo randomico quali sfere saranno già verdi, ovvero non necessiteranno di una cassa per cambiare colore, e quante rosse; durante questa scelta verrà riempito un array allo scopo di tenere in memoria il numero di casse ancora da riportare, questo array sarà passato alla classe navicella che si occuperà di verificare quando la partita dovrà finire.

La navicella e la ExtendedCamera
Vengono istanziate una classe navicella e una classe ExtendedCamera, le quali si occupano rispettivamente del movimento del soggetto del videogioco e della sua iterazione col mondo, e della gestione della telecamera che dovrà sempre seguire la navicella.

Lo scenario

Viene creato lo scenario della scena, inserendo nello scenario una mesh a cui sarà applicato un materiale che utilizza una texture rappresentante uno sfondo spaziale; questa, come tutte le mesh caricate precedentemente, sarà scalata in modo proporzionale al suo scopo.

L'animazione finale:

Viene poi generato tutto il necessario per definire l'animazione finale che sarà eseguita solo una volta completata la partita:

1. Un nodo di scena a cui associare la camera
2. Un'istanza della classe Animation, a cui verrà assegnato un nome, una durata in secondi e un tipo di interpolazione tra chiavi di animazione
3. Un nodo di scena per l'animazione
4. Una traccia che conterrà le chiavi di animazione
5. Delle chiavi di animazione che rappresentano le coordinate da cui dovrà passare l'oggetto dell'animazione
6. Un AnimationState che si occuperà dell'aggiornamento del nodo animato

PFrameListener

Il framelistener è la classe che si occupa di modificare gli oggetti presenti nella scena in relazione al tempo trascorso e agli input dell'utente:

Il passaggio degli input
La gestione degli input è quasi totalmente effettuata dalla classe Navicella, per questo il frame listener si occupa di richiamare a ogni frame il metodo update di questa classe passandogli come parametro gli oggetti di input tastiera e mouse che sono stati precedentemente creati.

La rotazione dello stargate
Il frame listener si occupa anche di far ruotare lo stargate nel centro della scena, per fare questo applica al nodo di scena padre di quella struttura un piccolo angolo di rotazione ad ogni frame.

L'animazione finale:
In caso di rilevata fine della partita il frame listener si occupa di :

- attaccare la camera al nodo precedentemente creato nell'application.cpp a questo scopo
- attaccare la navicella al nodo di animazione
- fare in modo che l'animazione proceda
- fare in modo che al termine dell'animazione l'applicazione si concluda

Overlay iniziali
Gli overlay iniziali vengono creati nel framelistener e utilizzati per visualizzare una piccola storia di introduzione al mondo di Asteroid Cargo, un riepilogo della situazione attuale e la lista dei comandi accettati dal gioco.

Gli overlay in-game
Abbiamo un unico Overlay gestito dal framelistener durante la fase di gioco: il timer. Questo conteggia i secondi passati dall'inizio della fase del gioco, fino al successo della missione, dove verrà bloccato (o meglio, non più aggiornato) visualizzando il tempo impiegato.

Il toggle
Il toggle "`m_GUIToggle`" serve ad assicurarsi che sia stato visualizzato un quantitativo di frame sufficienti per poter passare all'overlay seguente (o all'inizio del gioco).

Navicella

La classe navicella si occupa della gestione del sistema di movimento del videogioco, dall'implementazione degli effetti dei malus, e dall'implementazione delle routine per raccogliere e depositare le casse.

I limiti della scena
Per simulare il fatto di essere stati intrappolati in una regione di spazio, all'interno del metodo update della classe navicella, si è implementata una routine che verifica che il nodo di scena della navicella non abbia superato il limite delle 11000 unità su uno dei tre assi: se questo è successo allora il nodo viene riposizionato dalla parte opposta della scena sulla coordinata che ha superato il valore. In questo modo non è possibile allontanarsi troppo dal centro della scena.

Lo spherequery
Per fare in modo che si possano raccogliere le casse che sono vicine, è stata creata una SphereQuery la cui sfera, di raggio 300, viene ad ogni frame spostata per seguire il nodo di scena della navicella.
Alla pressione del tasto sinistro del mouse una routine restituisce una lista di oggetti che intersecano o sono all'interno della sfera, di questi viene poi preso il primo, eseguita l'animazione che fa scomparire la cassa e, a seconda del QueryFlag che possiede si attiverà un malus o si farà comparire una cassa al di sotto della navicella, il cui nodo sarà legato a quello della stessa, in modo da muoversi assieme.

Il RayQuery
Per depositare le casse e terminare la missione, si è implementata una RayQuery che viene eseguita alla pressione del tasto destro del mouse.
Questa query proietta un raggio dal nodo di scena dell'astronave nella direzione con cui questo è orientato, e restituisce gli oggetti intersecati dal raggio che hanno il Query Flag degli oggetti di missione, ovvero le sfere ancora rosse sul portale.
Se la query restituisce uno di questi oggetti, allora comincia l'animazione che fa sparire la cassa da sotto l'astronave, e al termine di questa la sfera colpita dal raggio cambia shader e diventa verde; inoltre viene aggiornato l'array che tiene traccia degli oggetti missione, in modo da poter stimare quando si è conclusa la partita.

La routine per il raddrizzamento
È stata implementata una routine per riposizionare l'astronave in modo da farle conservare l'orientamento sul piano xz, ma riportare a 0 l'orientamento sull'asse y. Questa funzione è richiamata ogni volta che si passa dallo stato di movimento di navigazione a quello di precisione.

Il Teletrasporto delle casse
Una volta che lo SphereQuery ritorna una cassa (ritorna un MovableObject, da cui deriviamo il nodo a cui è attaccato) attacchiamo i ParticleSystem ai due nodi interessati per simulare un effetto di teletrasporto e settiamo le variabili booleane di riferimento per impedire di prendere altre casse fin tanto che non ci siamo liberati di quella attuale e di portare avanti l'animazione nei frame a venire. L'animazione interessata è una scalatura del nodo posto davanti alla navicella.

Gli Overlay In-game

Navicella gestisce anche la parte riguardante la visualizzazione dei Malus attivi mediante overlay e informazioni testuali e il conto alla rovescia alla loro disattivazione. Per comodità abbiamo scritto il metodo `SetOverlay( String s, Real r ).`

Scelte Implementative

Scene Object

La classe SceneObject serve a definire qualsiasi oggetto della scena che potrebbe essere seguito e tracciato automaticamente dalla nostra camera. E' una classe generica che ci fornirà le fondamenta per creare la classe Navicella (che la estende).

Il sistema di puntamento della telecamera

Il sistema di puntamento della telecamera si basa su quattro nodi: il nodo della navicella, il nodo che identifica, ipoteticamente, quello che la navicella sta puntando, il nodo della camera e il nodo "ideale" della posizione della camera. Per effettuare un movimento fluido (sia del nodo della camera, che quello di visione), utilizziamo un vettore di traslazione moltiplicato per uno scalare, ovvero la nostra variabile di "durezza", compresa tra 0 e 1 (per ora posta di default a 0.09).

I Particellari

Si è fatto largo uso di ParticleSystem per ovviare alle difficoltà esistenti del renderizzare effetti simili con le tecniche convenzionali. I file .particle utilizzati nel progetto sono stati ottenuti partendo da una base costruita tramite l'ausilio del software Particle Editor, per poi essere parzialmente riscritti e personalizzati nel dettaglio. Possiamo notarli, in particolare, nelle scie dei motori e nell'effetto di teletrasporto per le casse.

Per comodità abbiamo scritto due metodi generici:

```
void    AttachParticle( SceneNode * n, ParticleSystem * p);
void    DetachParticle( SceneNode * n, ParticleSystem * p);
```

Il primo metodo attacca il ParticleSystem in argomento al nodo e imposta quest'ultimo come visibile. Il secondo metodo pulisce il buffer del ParticleSystem e lo stacca dal nodo.

Il sistema di movimento

L'intero sistema dei movimenti viene gestito da Navicella. Abbiamo definito due modalità di controllo, per una scelta di game design, e assegnato fattori a favore di una o dell'altra a seconda dell'operazione che si intende compiere. Meno genericamente, le nostre due modalità di controllo consistono in una con elevate velocità di spostamento, e l'altra con i comandi base per l'interazione col gioco.

Per la prima modalità abbiamo preferito prendere in prestito il modello dei controlli dai simulatori di volo: possiamo modificare Roll e Pitch per curvare la traiettoria, ma non Yaw (anche se in alcuni simulatori di volo è possibile). Nella seconda, invece, queste modalità spariscono per far posto ad un sistema di

controllo più preciso e meno rapido: avremo a disposizione la funzione di Yaw, di avanzamento, di retromarcia e di sali e scendi.

L'implementazione dei malus

Anche l'implementazione dei malus è stata fatta per una scelta di Game Design: volevamo complicare l'esperienza di gioco, senza però renderla stancante. Per far ciò, abbiamo creato dei Malus "ambigui" (come il turbo, che presenta vantaggi o svantaggi a seconda della modalità di controllo) e dei Malus "Maligni" (il maiale, crazy blindness, ecc…).