Savoca Enrico **26/05/2016** **Matricola: 1132805**

Progetto di Sistemi multimediali

Progetto 3 - Animazione multimediale sintetizzata

0. Scopo del gioco e comandi

Lo scopo del gioco è quello di muoversi all'interno di un labirinto per raggiungere degli allarmi (rappresentati da colonne verdi) e disattivarli tutti entro un tempo limite. Quando un allarme viene disattivato, il suo colore passa da verde a rosso.
Comandi:
w - Muovi avanti
s - Muovi indietro
a (oppure **q**)- Gira verso sinistra
d (oppure **e**)- Gira verso destra
Barra spaziatrice - disattiva allarme
Esc - esci dal gioco (quando viene visualizzata la schermata di vittoria o di Game Over)

1. Installazione, compilazione ed esecuzione

All'interno della cartella "progetto" si può trovare il file mazeEscape.cpp. La compilazione di questo sorgente, permette di generare un eseguibile (di estensione .exe in windows e .out in linux). Nella stessa cartella si possono inoltre trovare la cartella "texture", contenente le texture utilizzate (in formato bitmap), la cartella "audio", contenente le tracce audio utilizzate nel progetto e la cartella "code", in cui sono presenti le classi (in ulteriori file in formato .cpp) utilizzate dal file sorgente principale per generare l'ambiente. Si tratterà in maniera estensiva il contenuto della cartella "code" nel capitolo 3 del presente documento.
Perché la compilazione avvenga correttamente, è necessario che determinate librerie, grafiche e audio, siano installate.
Su linux è possibile installare le librerie d'interesse, tramite l'utilizzo del terminale invocando il comando

```
sudo apt-get install
```

seguito dal nome del pacchetto da installare.
Si dovranno installare i seguenti pacchetti per OpenGL: **freeglut3** e **freeglut3-dev**
(in alcune versioni di linux è inoltre necessario il pacchetto **binutils-gold**).
I pacchetti per OpenAL da installare sono invece **libghc-alut-dev** e **libalut-dev**.
Dopo l'installazione dei pacchetti, sarà possibile compilare i sorgenti. Ponendosi nella cartella contenente il file **mazeEscape.cpp**, si potrà invocare la compilazione, utilizzando i linker seguenti: -lGL -lGLU -lglut -lopenal -lalut.
Il comando da invocare da terminale, sarà qualcosa di simile a questo:

```
g++ mazeEscape.cpp -o mazeEscape.out -lGL -lGLU -lglut -lopenal -lalut
```

(in ambiente Windows, sostituire mazeEscape.out con mazeEscape.exe).

Dopo aver compilato il file, verrà generato un file col nome mazeEscape.out
Restando nella stessa cartella, per l'esecuzione del file sarà necessario invocare il comando

```
./mazeEscape.out
```

2. Requisiti del progetto soddisfatti

Tutti i punti esposti nella specifica del progetto sono stati implementati.
Il labirinto e la posizione degli allarmi vengono generati tramite la lettura del file **maze.txt**.
Il labirinto viene generato a partire da una matrice. Le dimensioni della matrice sono indicate nel file nella prima riga (numero righe) e nella seconda (numero colonne).
Segue quindi la struttura del labirinto, tramite una sequenza di 0 (spazio vuoto) e 1 (muro).
Successivamente è presente il numero di allarmi.
Seguono quindi le coordinate di ogni allarme (numero colonna e numero riga).
Lo scopo del gioco è quello di disattivare gli allarmi (delle colonne verdi), all'interno del labirinto entro un determinato tempo. Alla disattivazione dell'ultimo allarme e allo scadere del tempo, verrà visualizzato il record: se sarà stato disattivato ogni allarme, si mostrerà il tempo mancante, diversamente verrà mostrato il numero di allarmi non disattivati.
Per evitare che il giocatore esca dagli spazi del labirinto, si è deciso di aggiungere delle mura attorno all'intero labirinto definito dall'utente.

IMPORTANTE: perchè il gioco funzioni correttamente è necessario che la prima casella del labirinto (0,0) sia vuota (0), poichè il giocatore sarà collocato in quella posizione. Inoltre gli allarmi dovranno essere posizionati in degli spazi vuoti all'interno del labirinto e dovrà esistere almeno un percorso che permetta al giocatore di raggiungerli tutti.

Oltre ai punti fondamentali richiesti dalla specifica, sono stati implementati tutti i punti facoltativi.

Punti facoltativi soddisfatti

1. Presenza di *texture* per rendere la scena più realistica;
2. Gestione del Tempo nella barra del titolo;
3. illuminazione tramite l'effetto torcia e aggiunta dell'effetto nebbia;
4. Utilizzo di diversi materiali per gli oggetti nella scena;
5. Aggiunta di suoni per segnalare la presenza di un allarme in vicinanza.

Per rendere il gioco di più gradevole, sono state effettuate delle aggiunte.

Punti aggiuntivi soddisfatti

1. Presenza nella barra del titolo un indicatore dell'orientamento del giocatore all'interno del labirinto: Nort, Est, Sud, Ovest;
2. Indicatore nella barra del titolo del numero di allarmi da disattivare;
3. Aggiunta di un effetto audio di sottofondo per rendere l'atmosfera più cupa, migliorando così l'esperienza del giocatore.

3. Implementazione

Il file principale (mazeEscape.cpp) contiene il main e le istanze delle classi utilizzate per realizzare il labirinto e gestirne la logica, la grafica e la componente audio. In generale le istanze vengono inizializzate nel **main()** e si chiamano i loro metodi di aggiornamento, in base alle necessità, all'interno delle funzioni di callback registrate. Le funzioni di callback registrate, si occupano di aggiornare la grafica, la barra del titolo e di "ascoltare" le interazioni del giocatore con i dispositivi di input (es: tastiera).
Segue una lista delle classi implementate e del loro utilizzo.

3.1 maze

Contiene due classi: **alarm** e **maze**. La prima viene utilizzata per costruire gli allarmi all'interno del labirinto, tramite il costruttore. La seconda, **maze** si occupa della creazione e gestione del labirinto dal punto di vista logico. Il labirinto viene generato tramite il costruttore, a partire da un file chiamato "**maze.txt**". Ulteriori metodi presenti nella stessa classe si occupano di gestire le possibili collisioni tra il muro e il giocatore, la disattivazione degli allarmi da parte del giocatore e il controllo dello stato del gioco (se è stato vinto).

3.2 audioManager

La componente audio del gioco viene gestita all'interno di questa classe. All'inizializzazione, vengono generati dei buffer e delle sorgenti audio (*sources*). Ad ogni buffer è associato un file, mentre ogni sorgente ha delle proprietà come la posizione in cui si trova nello spazio, il suo volume (**gain**) e la sua velocità (**pitch**). Ogni sorgente viene associata ad un buffer e tramite un determinato comando (**alSourcePlay**), sotto certe condizioni, viene riprodotta la traccia audio. Oltre ad un rumore ripetuto per creare un'atmosfera cupa e realistica, sono presenti i suoni degli allarmi che si spengono nel momento in cui vengono disattivati.
La posizione dell'ascoltatore (listener) viene aggiornato ogni volta che viene ridisegnata la scena. L'alternativa sarebbe stata quella di aggiornare la posizione delle sorgenti, ma sarebbe stata più dispendiosa computazionalmente.

3.3 lightManager

Questa classe si occupa dell'inizializzazione dell'effetto nebbia e della torcia. La torcia manterrà la stessa posizione fin dall'inizio, come il giocatore. L'ambiente si sposterà e verrà illuminato di conseguenza. L'effetto nebbia viene creato tramite dei metodi ad hoc che ne determinano il colore, la densità e la velocità di attenuazione.

3.4 drawer

La classe **drawer** ha lo scopo di ridisegnare gli elementi della scena nel momento in cui si invocano determinati suoi metodi. La classe si occupa del disegno del labirinto, del pavimento, del soffitto e degli allarmi. Di questi ultimi, gestisce le proprietà grafiche da applicare nel disegno degli allarmi attivi o disattivati.

3.5 textureLoader

Una volta inizializzata, questa classe viene utilizzata per due metodi principali. Uno, **bmpLoader(...)**, viene utilizzato per il caricamento di texture a partire da file in formato bitmap. Il secondo metodo (privato) si occupa invece di fare il binding tra una texture

caricata e una variabile che la identifica, tramite la quale si gestirà all'interno dell'applicazione.

3.6 controller

All'interno di questa classe sono presenti i metodi tramite i quali viene gestito la logica interna del gioco. Tra i metodi più importanti, spiccano quelli relativi all'interazione dell'utente:

- **disableAlarm** : disattiva un allarme vicino;
- **makeAction** : compie un'azione in base al comando dell'utente;
- **updateState** : aggiorna la barra del titolo ciclicamente.

Nella barra del titolo, si mostrerà il tempo trascorso e mancante alla fine del gioco, il numero di allarmi da disattivare e l'orientamento del giocatore all'interno del labirinto.
Due metodi ulteriori setWon e setLost gestiscono la logica del gioco rispettivamente nel momento di vittoria o sconfitta. In entrambi i casi si mostrerà su schermo nero il risultato e il record.

PLUS:
Se si devono disegnare oggetti con texture e materiali diversi conviene fare un ciclo for per ogni materiale/texture e all'interno fare un ciclo for per disegnare il labirinto. In questo modo non si occupa il buffer della gpu per la texture (del labirinto se ne occupa la cpu)
In generale in realtà si usa un albero.