NonGravitar

Sviluppo di un gioco in C++

Realizzato da:

Gavanelli Sofia

Lena Erika

Ritrovato Martina

Indice

1. Introduzione

Il progetto consiste nella creazione di un gioco in C++, NonGravitar, rivisitazione del più famoso Gravitar rilasciato da Atari nel 1982.
Nel gioco originale il giocatore controlla una piccola astronave e si muove attraverso i pianeti per distruggere, sparandogli, i diversi bunker presenti su di essi.
Il gioco originale ha il nome Gravitar poiché la gravità gioca un ruolo importante in esso: la navicella viene lentamente tirata verso il basso se il giocatore non accelera; nella versione da noi realizzata ciò non accade, motivo per cui è stata chiamata NonGravitar.

2. Grafica

Riguardo la grafica, abbiamo preferito attenerci a uno stile minimal e "da terminale", usando però colori vivaci che risaltassero.
La libreria grafica usata è SFML (Simple and Fast Multimedia Library), scritta in C++, e si compone di diversi moduli:

- System, che contiene varie classi come vettori e stringhe Unicode
- Window, che gestisce gli input e la finestra
- Graphics, che si occupa della grafica in 2D, inclusi sprite, poligoni e rendering
- Audio, per la riproduzione e registrazione di audio
- Network

Noi abbiamo usato i moduli Window per gestire appunto le finestre del gioco, e Graphics per disegnare i pianeti, terreno, proiettili e la navicella.

3. Gameplay

3.1 Comandi

Il giocatore ha a disposizione vari pulsanti: due per ruotare la navicella a destra o sinistra, uno per accelerare e uno per avanzare normalmente, uno per sparare e, infine, uno per attivare il raggio traente.

3.2 Obiettivo

Nei vari pianeti il giocatore ha l'obiettivo di distruggere i bunker che gli sparano quando la navicella si avvicina, usando il raggio traente per raccogliere le taniche di carburante

sparse per il pianeta quando è necessario. Una volta che tutti i bunker verranno distrutti, il pianeta scomparirà. Quando poi tutti i pianeti verranno distrutti, il giocatore si sposterà in un nuovo sistema solare e continuerà a giocare.

Il giocatore perderà una vita se si scontrerà con il terreno, mentre la sua barra della vita calerà quando colpito dai nemici. Se invece finirà il carburante, la partita terminerà.

3.3 Implementazione

CLASSI

Il gioco è strutturato in classi:

- ★ Menu: gestisce l'interfaccia iniziale, le informazioni sul gioco, il game over e la classifica finale.
- ★ Game: avvia il ciclo principale che processa gli eventi del gioco, detta lo scorrere del tempo e gestisce la pausa quando chiamata dall'utente.
- ★ Planet: rappresenta un pianeta.
- ★ Terrain: rappresenta il terreno di gioco interno ad un pianeta.
- ★ Spaceship: gestisce la navicella controllata dal giocatore.
- ★ Bunker: oggetti disposti sul pianeta, che il giocatore deve distruggere per accumulare punti.
- ★ Tank: taniche di carburante sparse per i pianeti con diverse capacità.
- ★ Bullet: classe per la gestione dei proiettili.
- ★ TextBox: gestisce tutte le didascalie utili alla comprensione e all'estetica del gioco.

3.3.1 Menu

La classe menu gestisce le varie interfacce che possono essere visualizzate durante il gioco: il menù principale, la spiegazione delle istruzioni, la conclusione del gioco e la classifica finale. Inoltre, essa gestisce l'inizio delle varie partite: tramite la funzione initialize() crea una nuova istanza di game e chiama su di essa la funzione run(), a questo punto il gioco può iniziare. Quando questo finisce (per game over o per chiusura della finestra) un valore viene ritornato al menù: se è -1 viene chiamata la funzione saveRank() che scarica la classifica aggiornata su file, mentre se è un valore diverso questo viene salvato come punteggio del giocatore e viene preso poi in esame nella funzione createRank() per aggiornare la classifica.

3.3.2 Game

La classe game gestisce la chiamata di tutte le altre classi; all'interno di essa, infatti, vengono dichiarati tutti gli oggetti necessari al gioco: una navicella, un vettore di pianeti e tutte le variabili secondarie. La funzione run() rimane in funzione finché l'utente non

finisce le vite (gestito dalla variabile alive) o chiude la finestra (variabile close), quando accade una delle due opzioni il gioco viene interrotto. In questa funzione è gestita anche la pausa, infatti, l'update() e il render() della finestra avvengono solo se il gioco non è in pausa, in caso contrario viene visualizzata la scritta "PAUSE" e il gioco si interrompe. Infine, tiene conto dei pianeti che non sono ancora stati distrutti e, quando la variabile planetsLeft diventa zero chiama la funzione newSolarSystem() per permettere all'utente di continuare a giocare in un nuovo sistema solare.
Nella funzione di update viene gestita anche l'entrata/uscita della navicella da un pianeta e la vista relativa (all'interno del pianeta la vista si muove con navicella), la funzione si occupa infatti di controllare se la navicella collide con un pianeta (dall'esterno) o se ne esce, nel caso si trovi all'interno, allontanandosi dalla sua superficie o distruggendo tutti i bunker che vi sono collocati.Allo stesso modo la funzione di render() disegna il sistema solare quando la navicella non è dentro ad un pianeta ed il pianeta con il relativo terreno quando la navicella si trova all'interno di esso.

3.3.3 Planet

La classe pianeta gestisce tutto ciò che avviene all'interno del pianeta; essa contiene al suo interno un oggetto terreno e un vettore di taniche e di bunker, i quali vengono posizionati sulla linea del suo terreno dalla funzione setObject(). La funzione update() prende in input per riferimento la navicella ed il punteggio, da aggiornare, e si occupa di gestire le collisioni di proiettili, navicella e raggio traente con il terreno, nonché di controllare per ciascuno dei bunker presenti se i proiettili sparati hanno colpito la navicella. Infine, si occupa di decrementare la vita dei bunker o eliminarli, quando vengono colpiti dalla navicella.

Il file Object.h definisce una classe astratta ereditata dalle classi 'bunker' e 'tank' così da poter gestire queste ultime in un unico vettore di 'oggetti' all'interno delle classi planet e terrain, collocandoli in maniera randomica sul terreno.

3.3.4 Terrain

La classe terreno si occupa della rappresentazione interna del suolo del pianeta, costituito da un array di vertici determinati in maniera randomica. La classe è poi dotata di un metodo che viene invocato dalla classe pianeta e si occupa di collocare, impostando posizione e rotazione in relazione all'inclinazione del terreno, gli oggetti del pianeta (bunker e tank). Il metodo collision(), anch'esso invocato da planet, prende in input la posizione dell'oggetto del quale si vogliono controllare le collisioni con il terreno e restituisce la distanza di tale punto dal segmento di terreno sopra il quale si trova l'oggetto.

3.3.5 spaceShip

La classe spaceship è dotata di un metodo che gestisce le funzionalità chiamate dall'utente impostando di volta in volta i parametri (IsMoving, IsAccelerating, IsRotating, IsFiring, RaggioT) della navicella a true (quando un tasto viene premuto) o a false (quando un tasto viene rilasciato).
La navicella possiede poi un vettore di proiettili (che come nel caso dei bunker, viene riempito dalla funzione recharge()) ed altri parametri per la gestione del raggio traente, della vita e del carburante (i cui valori sono esposti in alto a sinistra sulla schermata di gioco).

3.3.6 Bunker

La classe bunker, così come la navicella, è dotata di variabili che gestiscono i parametri vitali e di un vettore di proiettili.
Esistono tre sottoclassi della classe bunker: redBunker, yellowBunker e bluBunker; ciascuna di esse ha un suo metodo di attacco e una diversa "longevità". I bunker si attivano nel momento in cui la navicella si avvicina e smettono di sparare quando essa si allontana. Ognuna delle tre sottoclassi ha un proprio costruttore e sovrascrive i metodi di ricarica (recharge()) e di rappresentazione (draw()) del bunker.

3.3.7 Tank

I serbatoi sono rappresentati da semplici rettangoli con una textBox per identificarli e mostrarne la quantità. Quando la navicella attiva il raggio traente in loro prossimità, l'update del pianeta invoca la funzione toEmpty(), che "svuota" le taniche cambiando il colore e impostando la loro quantità a zero.

3.3.8 Bullet

La classe bullet ha solo due parametri: una 'rectangle shape' che ne costituisce la rappresentazione ed un parametro di direzione che identifica la retta sulla quale esso si dovrà muovere. I proiettili vengono spostati sullo schermo dalla funzione fire(), che può essere chiamata con o senza parametri; nella versione fire(int x, int y) la direzione in cui vengono sparati i proiettili viene decisa al momento della chiamata della funzione (es. nel caso dei redBunker che sparano in direzione della navicella a seconda di dove essa si trova). Invece, fire() è utilizzata dagli altri bunker e dalla navicella poiché questi possiedono già una direzione fissata al momento della creazione. Infine, i proiettili possono essere spostati ad una velocità maggiore nel caso in cui l'oggetto dal quale vengono sparati si stia nel frattempo muovendo.

3.3.9 textBox

Attraverso la classe textBox è stato possibile creare e gestire tutte le scritte che aiutano la comprensione del gioco, dal titolo e le istruzioni alla visualizzazione della vita, dei punti e delle capacità delle taniche ('F 2L' o 'F 4L'). Le funzioni di textBox ne gestiscono le caratteristiche principali (posizione, grandezza, colore del testo, font) e la visualizzazione (draw()). Quest'ultima può essere di due tipi: drawRelative() prende in input la posizione rispetto al centro della view per fare in modo che determinate scritte si muovano insieme al movimento della vista (il punteggio e la vita sono, infatti, sempre in alto a sinistra, indipendentemente dal movimento della view, che avviene in base al movimento della navicella). La draw(), invece, disegna semplicemente il textBox basandosi sulle caratteristiche già impostate e serve invece per le scritte che hanno una posizione statica (come ad esempio tutte quelle gestite in menu).