surviveX

Alessandro Capriccioni #000832273

Francesco Licchelli #0001041426

Nico Wu #0001028979

09/06/2022

Sommario

Questo gioco è stato creato per il progetto del corso di Programmazione, è scritto in linguaggio C++ ed il principale scopo è di imparare a lavorare in gruppo, di esercitarci in direzione di quello che succede nel mondo reale, oltre all'ampliamento delle nostre conoscenze del C++.

1 Strumentazione

Per la realizzazione del progetto sono state utilizzati i seguenti strumenti:

- Ubuntu 21.04 – Sistema Operativo utilizzato per lo sviluppo del progetto
- GitHub – Strumento per la gestione delle versioni e condivisione del codice
- Visual Studio Code – IDE per programmare in C++
- g++ v.9.4.0 – Compilatore codice C++
- GNU Make 4.3 – Strumento per la generazione dell'eseguibile
- GNU gdb v.12.0.90 — Debugger codice C++
- Cppcheck v.2.3 — Strumento per l'analisi statica del codice
- Discord – Per discussione e organizzazione fasi progetto

2 Struttura degli oggetti nel programma

```
root
├── main.cpp...  Istanza un oggetto della classe SurviveX.
├── Fighting...  Contiene le classi interessate al combattimento,
│   ├── Enemy      quali Player ed Enemy, derivate da Shooter, e la classe Weapon
│   ├── Player
│   ├── Shooter
│   └── Weapon...  Classe per la gestione del combattimento a distanza.
├── Lists...    Contiene una serie di classi utilizzate per la
│   ├── AList     gestione degli oggetti presenti nei livelli.
│   ├── EList
│   ├── KList
│   └── PList
├── MapManagement...  Contiene le classi per la gestione dei livelli, le
│   ├── Level           interazioni tra oggetti e la rappresentazione per l'utente.
│   ├── LevelNet
│   ├── LevelTemplate
│   └── Map
├── Objects...  Contiene le classi degli oggetti che possono essere trovati
│   ├── Ammo      nei livelli.
│   ├── Artifact
│   ├── Buff
│   ├── Key
│   └── Portal
└── misc...         Contiene le classi ausiliarie, la classe delle costanti.
    ├── data...         Contiene i layout delle stanze e il punteggio per ogni
    │   ├── map          partita giocata
    │   └── scoreboard
    ├── headers...         Contiene le dichiarazioni delle costanti, delle stringhe
    │   ├── constants.hpp    delle strutture e delle funzioni ausiliarie usate dal
    │   ├── game_str.hpp     gioco.
    │   ├── library.hpp
    │   └── structures.hpp
    ├── Coord...       Classe ausiliaria per la gestione delle coordinate.
    ├── SurviveX...      Classe principale del gioco.
    ├── c_ncurses.cpp...  Modifiche a funzioni ncurses per compatibilità con Coord.
    ├── c_ncurses.hpp
    └── utils.cpp...    Funzioni ausiliarie.
```

3 Requisiti minimi

Il **progetto** richiede la creazione di un gioco in grafica ASCII, sviluppato su più livelli, con un punteggio della partita giocata e senza traguardi, l'unico modo per terminare il gioco è nel caso in cui il protagonista perda tutta la vita.

Il **protagonista** viene controllato dal giocatore tramite tastiera, ha tre cuori e ogni cuore è composto da dieci punti, durante la partita può incontrare nemici che tolgono cuori, artefatti per aprire le porte e poteri per vari effetti.

Gli **artefatti** aumentano la vita mentre i **poteri** sbloccano passaggi per altre stanze e ripristinano la vita se in precedenza colpiti da un nemico.

I **nemici** sono di diverso tipo e controllati dal computer, possono muoversi e sparare togliendo cuori al protagonista.

La **mappa** ha una visuale dall'alto ed è composta da infinite stanze con più ingressi/uscite che il protagonista può visitare; per passare da una stanza all'altra il protagonista deve prendere artefatti, inoltre deve trovare una stanza nello stesso stato in cui l'ha lasciata, cioè con gli stessi nemici, artefatti e stato delle porte presenti nella sua ultima visita.

I **vincoli** nello sviluppo del progetto sono i seguenti, gli elementi del progetto come le stanze, i nemici, i poteri e gli artefatti vanno gestiti con strutture dati dinamiche includendo quindi inserimento e rimozione, inoltre bisogna organizzare il progetto in più file usando le classi.

4 Progettazione e divisione del lavoro

Inizialmente abbiamo stilato un riassunto con tutte le caratteristiche e i vincoli che il progetto doveva avere, abbiamo visto giochi simili in ASCII da cui prendere spunto e infine abbiamo organizzato il lavoro in step, più precisamente ad ogni incontro si decideva quale fosse il prossimo aspetto o richiesta del progetto da sviluppare e si divideva il lavoro per far funzionare quell'aspetto.
Avendo adottato questo approccio incrementale nello sviluppo del codice non c'è una netta distinzione nel lavoro dei diversi membri, il lavoro di un membro completava quello di un altro.
Ad esempio nello spawn degli oggetti nella stanza un membro si è occupato del protagonista, un altro dei nemici e l'ultimo degli artefatti, per poi nell'incontro successivo aggiornare il codice su GitHub con le nuove funzioni e risolvere eventuali bug.

5 Scelte implementative e descrizione

Per tutto lo sviluppo del gioco si è fatto riferimento a uno stile di programmazione sia procedurale e sia orientato ad oggetti. Per quanto riguarda la progettazione del gioco abbiamo adottato un ibrido tra l'approccio top-down e bottom-up. Durante ogni incontro settimanale si discutevano le varie implementazioni e le eventuali ottimizzazioni del codice, tra le più importanti scelte implementative di ciascuna cartella vi sono:

- **Fighting** è la cartella principale contenente le varie classi interessate al combattimento, tra le scelte più importanti ci sono le implementazioni della classe Weapon,

della superclasse Shooter e delle sue due classi derivate Player ed Enemy.
Weapon è la classe che contiene tutte le informazioni che riguardano le armi da sparo, nella fase di progettazione abbiamo progettato tre tipologie di armi che sparano proiettili a velocità, danno e gittate diverse. Tra le operazioni più importanti di questa classe citiamo la resa del proiettile e la collision detection entrambe implementate nella funzione shoot:
Per lo sparo di un proiettile viene valutato il tempo di delay per limitare la frequenza di sparo dello shooter; a partire dal chiamante viene stampata l'icona del proiettile (la quale varia in base alla direzione) finché non arriva alla sua gittata massima oppure non incontra un ostacolo e ritorna un intero per segnalare se il proiettile è andato a segno, in caso affermativo Map toglie vita al bersaglio.
Durante lo sviluppo si è deciso di implementare la classe astratta **Shooter** per integrare le classi **Player** e **Enemy** e far riuso del codice. Gli attributi di Player vengono salvati in modo sequenziale in un array di interi per alleggerire il codice, mentre per quanto riguarda l'implementazione di Enemy, si è deciso di implementare quattro tipologie di nemico, ovvero il nemico base, con maggior attacco, con maggior vita e boss.

- **Lists** contiene le strutture dinamiche per salvare gli oggetti presenti in ciascun livello; vengono descritte le operazioni sulle liste per i diversi oggetti quali: inserimento in testa, la rimozione e la ricerca.

- **MapManagement** è una delle cartelle più importanti del programma, in essa si trovano le diverse classi per la gestione dei dati, il salvataggio dei livelli e la grafica del gioco.
  Nella fase iniziale del gioco la classe **LevelTemplate** interpreta e genera prototipi dei livelli dal file di testo /data/map per ottimizzare l'utilizzo della memoria RAM durante l'esecuzione del programma.
  La classe **LevelNet** è la classe designata al salvataggio e all'indicizzazione dei diversi livelli, per fare ciò viene implementata una bilista di biliste, si tratta di una scelta implementativa mirata per ottenere la circolarità delle stanze e poterne indicizzare un numero non definito a priori. Nella "rete" di livelli ogni livello è salvato come un nodo con una coordinata ad esso associato, usando tale sistema è stato possibile collegare tutti i nodi che sono adiacenti tra loro (circolarità).
  La classe **Level** invece è la classe che contiene tutti dati che caratterizzano un livello, ovvero un LevelTemplate ed una serie di liste di oggetti (portali, chiavi, artefatti, buff, nemici), in essa si trovano le funzioni riguardanti l'inizializzazione delle varie liste, ed è anche la classe principale che cura l'aspetto grafico del gioco, facendo visualizzare il vettore di caratteri del LevelTemplate con l'aggiunta degli oggetti della stanza corrente.
  Infine la classe **Map** è il principale gestore del gioco. Essa si occupa delle interazioni tra protagonista e gli artefatti nei livelli, del movimento e dello sparo del protagonista e dei nemici, inoltre completa la grafica del gioco aggiungendo la stampa degli eventi.

- **Objects** contiene le classi che definiscono i campi e alcuni metodi elementari degli oggetti della stanza.

- **misc** è la cartella contenente le classi ausiliarie, i dati per la configurazione dei templates delle stanze, l'header delle costanti e la classe principale del gioco.
  SurviveX è la classe principale che gestisce il menu iniziale del gioco, la stampa della leaderboard alla fine del gioco ed elabora gli input presi dall'utente.
  Poiché tutte le classi del gioco utilizzano le coordinate, è stata creata una classe che incorpori in sé tali operazioni, per questo motivo è stata creata la classe **Coord**.

6 Relazione fra classi

Per passare dal file contenente la mappa a ciò con cui il giocatore interagisce il gioco si basa su tre livelli di astrazione:

1. Lettura da file di testo e creazione lista di LevelTemplate:
   durante la creazione della mappa, viene letto carattere per carattere quanto presente nel file, il quale contiene un rigo per ogni LevelTemplate. I caratteri letti vengono interpretati e vengono salvati i corrispondenti caratteri in un vettore attributo del LevelTemplate che sta caricando in quel momento.

2. Generazione livelli pseudocasuali:
   quando bisogna creare un livello si sceglie un LevelTemplate casuale. A questo, in posizioni casuali, si aggiunge un numero casuale di oggetti, garantendo però la presenza di almeno due nemici.

3. Stampa a schermo del livello:
   per la stampa del livello viene mostrato il contenuto dell'array caricato al punto 1. con l'aggiunta degli oggetti generati al punto 2. Inoltre, se nel livello sono presenti dei portali aperti, ogni nodo che li compone verrà rappresentato da uno spazio, altrimenti ogni nodo del portale sarà rappresentato da X.

7 Conclusioni e considerazioni generali

Il progetto è stato completato in una sua prima versione, che a nostro avviso è sicuramente ampliabile e migliorabile dalle "fondamenta" che abbiamo costruito; possono essere aggiunti nuovi tipi di nemico, con più scudo, più velocità, possono essere aggiunti artefatti con nuovi effetti e molto altro. Inizialmente abbiamo riscontrato delle difficoltà nella comprensione di alcune richieste del progetto risolvendo il problema con una mail al referente del progetto, fatto molto realistico nel mondo del lavoro; altro aspetto problematico è stato nell'incontrarsi solo online non avendo sempre una chiara comunicazione. Avendo iniziato con molto anticipo rispetto alla data di consegna non abbiamo avuto problemi né di tempistiche né nel trovare giorni liberi per gli incontri.

SCREEN RECORDING ESECUZIONE GIOCO

Glossario

file Documento di testo contenente le specifiche di ogni template. 3–5

giocatore Utente che utilizza il programma. 3, 5

gioco surviveX. 1, 3–5

protagonista Oggetto controllato dal giocatore. 3, 4