

Eternal Run: sviluppo di un gioco in C++ con la libreria ncurses

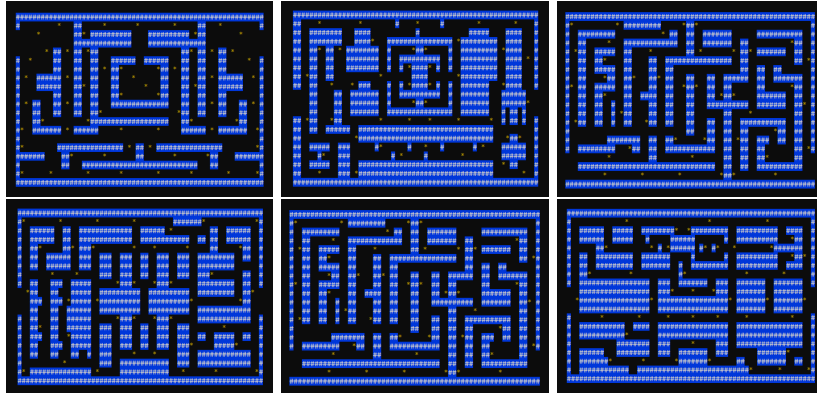
Matteo Lombardi
Enis Brajevic
Lorenzo Molinari
Christian Orsi

1 Introduzione

Eternal Run è un videogioco sviluppato in linguaggio C++ che utilizza la libreria ncurses per creare un'interfaccia giocabile tramite terminale. Il giocatore interpreta il protagonista che deve correre all'interno di mappe simili a dei labirinti, raccogliere monete, sconfiggere i nemici e raggiungere il prossimo livello.

2 Funzionamento del gioco


Il gioco presenta una rappresentazione grafica in formato ASCII ed è composto da un totale di 6 mappe, che vengono ripetute con un ordine casuale durante il gioco.



Le 6 mappe di Eternal Run.

L'obiettivo del giocatore, che interpreta il protagonista, è quello di raccogliere monete, sconfiggere i nemici e raggiungere il prossimo livello. Il gioco prevede anche livelli mercato tramite i quali sarà possibile aumentare le capacità dell'eroe,

rendendolo più forte e preparato a fronteggiare le sfide del labirinto. Tra i vari potenziamenti acquistabili, troviamo l'aumento della quantità di vite, fino ad un massimo di 3, l'incremento del valore dello scudo, l'acquisizione di una pistola, con la quale l'attraversamento dei livelli diventerà più facile, l'aumento della velocità dei proiettili, che può essere potenziata una sola volta, e il danno causato da essi sui nemici, i quali dispongono di una quantità di vita basata sul loro livello.



- Pozione vita (+1)	30M
- Pozione scudo (+1)	40M
- Pistola	50M
- Velocita' proiettili	100M
- Danno proiettili	120M

Livello precedente ->

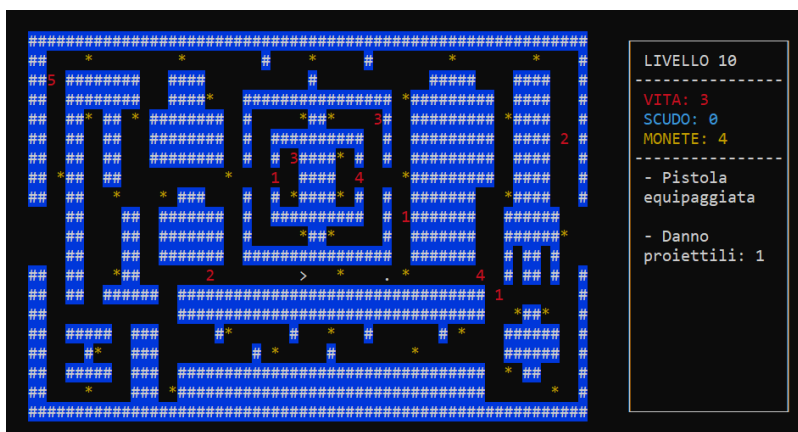
Livello mercato.

L'interfaccia grafica è composta da due finestre: una è quella principale, in cui vengono mostrate le mappe con le monete, i nemici e il protagonista che deve attraversare il labirinto; l'altra è la finestra delle informazioni, in cui sono presenti il livello corrente, le statistiche attuali, tra cui vita, scudo e monete, e varie informazioni sui potenziamenti ottenuti.

Al contatto con i nemici il protagonista perderà una vita e sarà costretto a ricominciare il livello con una nuova mappa generata in maniera casuale. Tuttavia, è possibile evitare questa situazione acquistando all'interno del mercato degli scudi, i quali permettono di entrare in contatto con i nemici senza subire perdite di vite.

Se vengono perse tutte le vite, non sarà possibile continuare e apparirà una schermata di game over, dalla quale sarà possibile decidere se abbandonare il gioco o iniziare una nuova partita.

Il giocatore ha inoltre la possibilità di tornare ai livelli e ai mercati precedenti attraversando le porte d'ingresso. Questo permette al protagonista di raccogliere le monete e di sconfiggere i nemici rimasti, al fine di aumentare la propria disponibilità economica sul mercato.



Esempio in-game.

3 Istruzioni di gioco

All'inizio di ogni livello, il giocatore viene posto all'interno della porta d'ingresso. Per avviare il gioco, è necessario cliccare sulla freccia direzionale corrispondente alla direzione in cui si trova il protagonista inizialmente. Una volta attivato il gioco, sarà possibile utilizzare le 4 frecce direzionali presenti sulla tastiera per muovere il personaggio all'interno della mappa.

Una volta acquisita la pistola dal mercato, sarà possibile utilizzarla sfruttando come grilletto la barra spaziatrice, permettendo al protagonista di sparare ai nemici incontrati lungo il percorso.

Inoltre, il giocatore ha sempre la possibilità di uscire dal gioco in qualsiasi momento premendo il tasto esc. Questo farà apparire una schermata che chiederà conferma della volontà di uscire, garantendo che il giocatore non perda accidentalmente il proprio progresso.

Nel mercato e nei vari menu del gioco, è inoltre possibile navigare tra le varie voci utilizzando le frecce direzionali up e down, e selezionare le opzioni desiderate premendo il tasto enter.

4 Scelte di implementazione

Il videogioco è organizzato all'interno di una cartella nella quale troviamo diversi file e sottocartelle.

Tra questi vi è un file README, il quale fornisce una breve descrizione riguardante l'utilizzo e l'esecuzione del videogioco.

Troviamo inoltre un makefile che consente di compilare il codice sorgente utilizzando un semplice comando da terminale "make". In questo modo verrà creato

un file eseguibile con nome "eternal_run", il quale conterrà il gioco completo che potrà essere avviato facendo doppio click sull'eseguibile oppure attraverso il comando da terminale `./eternal_run`.

È presente una cartella chiamata "storing_files", la quale verrà utilizzata per memorizzare i file di salvataggio generati durante il gioco. Questi file permetteranno di tornare ai livelli precedenti e continuare la partita da dove è stata interrotta.

Infine, all'interno dei file di gioco troviamo la sottocartella "src", che contiene tutto il codice sorgente suddiviso in file individuali. Questi sono organizzati in modo da garantire una struttura ordinata e una facile comprensione del codice. La struttura è la seguente:

- **main.cpp**

Nel file main.cpp del gioco, è stato mantenuto solo il metodo main() per conservare la leggibilità e la facilità di manutenzione del codice.

Tale metodo si occupa di preparare la libreria ncurses, che fornisce funzionalità di input/output testuali per l'interfaccia del gioco, e di mostrare lo splash screen, ovvero la schermata iniziale del gioco. Successivamente vengono create le 2 finestre di gioco e avviato il game loop, che rappresenta il cuore del gioco stesso. I metodi utilizzati per fare ciò verranno spiegati in seguito visto che la loro dichiarazione è presente all'interno degli altri file.

- **graphics.hpp e graphics.cpp**

Le interazioni con la libreria grafica ncurses vengono gestite attraverso i file graphics.hpp e graphics.cpp.

Qui troviamo le costanti globali e i metodi che permettono di interagire con la libreria ncurses. In particolare, le costanti comprendono quelle che riguardano gli indici dei pair, ovvero le combinazioni di colori utilizzate all'interno dell'applicazione. Queste costanti sono fondamentali per garantire che i colori utilizzati siano coerenti durante l'esecuzione del software. Inoltre, vi sono anche costanti che descrivono gli indici delle opzioni del menu mercato, permettendo una facile gestione delle opzioni offerte all'utente.

I metodi presenti nei file sono suddivisi in quattro sezioni principali.

La prima sezione include metodi generali, come quello per la creazione dei colori, che sono fondamentali per la corretta visualizzazione grafica all'interno del software.

La seconda sezione riguarda lo standard screen, in cui sono presenti funzioni che lavorano con l'input/output all'interno di esso. Tra queste troviamo, ad esempio, quella che mostra lo splash screen, ovvero la schermata iniziale del software.

La terza sezione include metodi che hanno a che fare con il game screen,

ovvero la schermata principale in cui vengono visualizzate le mappe, il giocatore, i nemici e i proiettili, oltre al livello mercato, dove gli utenti possono acquistare potenziamenti. Molti di questi metodi lavorano ciclando una lista di oggetti di classe Entity e controllando la loro attuale posizione x e y per visualizzarli correttamente a schermo. Vi sono anche metodi che permettono la visualizzazione a schermo delle schermate di uscita e di game over, una volta richiamati.

Infine, la quarta e ultima sezione include metodi che interagiscono con la finestra delle informazioni. Questi, molte volte, effettuano il refresh a schermo delle statistiche per garantire che siano sempre aggiornate ai valori attuali. Ad esempio, se il giocatore raccoglie una moneta, occorre effettuare il refresh della finestra al fine di mostrare l'aggiornamento del numero di monete in possesso.

- **entities.hpp e entities.cpp**

I file entities.hpp e entities.cpp sono fondamentali per la gestione degli elementi dinamici del gioco, ovvero le Entities.

Questi file contengono costanti globali, classi, strutture e metodi che consentono di gestire gli oggetti che hanno la possibilità muoversi all'interno della mappa.

Le costanti globali sono utilizzate per identificare le quattro direzioni possibili per le Entities, ovvero destra, sinistra, alto e basso. Queste costanti sono essenziali per il corretto funzionamento dei loro movimenti.

La gestione delle classi è stata implementata attraverso un sistema di sottoclassi:

La classe madre "Entity" consente di creare generici oggetti capaci di muoversi nella mappa e presenta gli attributi di posizione y e x, oltre all'attributo che indica la direzione dell'entity. Questa classe è stata utilizzata come base per la creazione di due sottoclassi, ovvero Player e Enemy. La classe Player rappresenta il protagonista del gioco e contiene al suo interno gli attributi e i metodi specifici per la gestione dei potenziamenti e delle azioni del giocatore.

La classe Enemy consente di creare i nemici e presenta gli attributi e i metodi specifici per la gestione della loro potenza e delle interazioni con il giocatore.

Infine, i file entities.hpp e entities.cpp contengono anche due strutture che consentono di creare liste utili alla gestione dei nemici e dei proiettili, con metodi che ne permettono l'aggiunta e la rimozione.

- **map.hpp e map.cpp**

I file map.hpp e map.cpp sono importanti componenti del sistema di gestione delle mappe all'interno del gioco. Essi contengono costanti globali, strutture e metodi che servono a rappresentare e gestire le 6 mappe presenti nel gioco.

Una singola mappa è rappresentata da una struttura di nome "map" che contiene un array bidimensionale 20x60, utilizzato per memorizzare il contenuto di ogni singolo blocco della mappa, oltre alle posizioni dell'entrata e dell'uscita, fondamentali per passare da un livello a quello successivo o precedente.

Il contenuto dell'array viene riempito con valori numerici che vanno da -1 (blocco vuoto) a 3, al fine di rappresentare blocchi entrata, uscita e muri. L'utilizzo di un array e non di una lista per memorizzare i blocchi è stato scelto per migliorare le prestazioni del gioco, poiché ogni volta che un'entità si muove all'interno della mappa, sarà sufficiente controllare il prossimo blocco nell'array tramite il suo indice, composto da posizione y e x, per determinare se è possibile muoversi, senza la necessità di cercare tramite un ciclo il blocco corrispondente all'interno di una lista.

Oltre a "map", troviamo anche una struttura utilizzata per costruire una lista di monete, che possono essere raccolte dal giocatore.

I metodi presenti in questi file comprendono la creazione delle 6 mappe e delle 6 liste di monete, nonché metodi più specifici per l'aggiunta o la rimozione di singoli blocchi e monete dalla lista.

- **storing.hpp e storing.cpp**

I file storing.hpp e storing.cpp gestiscono la memorizzazione dello stato dei livelli precedenti, permettendo così al giocatore di tornare a questi passando attraverso l'entrata della mappa.

Questa funzionalità è resa possibile grazie alla cartella "storing_files", che viene creata al passaggio al livello successivo se non esiste ancora. All'interno di questa cartella vengono salvati tre file diversi per ogni livello, ciascuno dei quali contiene informazioni specifiche sulle monete, sui nemici e sulla mappa corrispondente.

Il primo file contiene la posizione di ogni moneta rimasta nella mappa. Il secondo memorizza il livello dei nemici rimasti nella mappa, mentre la loro posizione non viene salvata in quanto vengono reinseriti in una locazione casuale nel momento in cui si torna al livello precedente. Infine, il terzo file contiene l'indice della mappa corrispondente al livello, il quale permette di prelevarla correttamente dall'array composto dalle 6 mappe del gioco presente nel file game.cpp.

I due file "storing" contengono diversi metodi, tra cui la creazione della cartella e dei file di storage, l'accesso alle informazioni contenute in essi e l'eliminazione di tutti i file una volta che il gioco viene chiuso o viene raggiunto il game over. Questi metodi forniscono una soluzione efficiente e ben organizzata per la gestione dello stato dei livelli precedenti, rendendo possibile per il giocatore di riprendere la propria avventura da dove l'ha

lasciata.

- **game.hpp e game.cpp**

I file `game.hpp` e `game.cpp` rappresentano la spina dorsale del progetto, poiché contengono tutti i metodi che consentono al gioco di funzionare correttamente mediante la connessione di tutti i componenti già menzionati.

In questi file vengono salvate le variabili globali che registrano i progressi del gioco, come quella del giocatore, della lista dei nemici e dei proiettili, delle 6 mappe del gioco e di altre variabili relative alle statistiche.

I metodi presenti includono quelli per creare una nuova partita, ricaricare un livello dai file di storing, generare nuovi livelli casualmente, gestire il movimento del protagonista, dei nemici, dei proiettili e la loro collisione.

Il metodo più importante in questi file è quello che gestisce il game loop, poiché "dà vita" al gioco. Questo metodo è costituito da un ciclo `while` che viene eseguito fino a quando l'utente esce dal gioco. Una variabile booleana viene utilizzata come guardia di questo ciclo. All'interno di esso, se l'utente ha fornito un input, quest'ultimo viene salvato in una variabile che viene controllata tramite uno `switch` per determinare la prossima mossa da eseguire, come ad esempio quella di cambiare direzione al protagonista oppure sparare.

In seguito, vengono aggiornate le posizioni di tutte le entità e infine viene chiamato `napms()` per fermare l'esecuzione del programma per diversi millisecondi al fine di ottenere un movimento regolare e gestibile dal video-giocatore.

```
while (!exit_game) {
    char input_char = wgetch(game_win);

    if (input_char != -1) {
        switch (input_char) {
            case KEY_DOWN:
                // move down
                break;
            case KEY_UP:
                // move up
                break;
            ...
            case KEY_SPACEBAR:
                // shoot
                break;
            case KEY_ESC:
                exit_game = true;
                break;
        }
    }
}
```

```

    }

    // move entities and refresh

    napms(80);
};

```

Codice semplificato del game loop

- **utils.hpp e utils.cpp**

I file `utils.hpp` e `utils.cpp` contengono metodi che svolgono compiti secondari ma che contribuiscono a mantenere il codice ordinato e facilmente gestibile.

Tra questi metodi troviamo quelli che hanno il compito di svolgere controlli sui movimenti delle Entities, come quello per la possibilità di muoversi all'interno di un determinato blocco o quello del calcolo della prossima posizione di un determinato oggetto.

Vi sono inoltre metodi che consentono di effettuare la pulizia dei puntatori all'interno delle liste, in modo tale da liberare la memoria utilizzata da esse una volta che non sono più necessarie, evitando che si accumulino in memoria. Questo contribuisce a mantenere la memoria del sistema libera e disponibile per altri compiti.

5 Divisione del lavoro

- **Matteo Lombardi:** interazioni con la libreria grafica, creazione delle mappe, implementazione delle entità, implementazione delle dinamiche di gioco, scrittura della relazione.
- **Enis Brajevic:** salvataggio dello stato dei livelli precedenti, creazione delle mappe, implementazione delle entità, implementazione delle dinamiche di gioco, scrittura del README.
- **Lorenzo Molinari:** creazione delle entità (giocatore, nemici e proiettili).
- **Christian Orsi:** creazione del livello mercato, implementazione dei potenziamenti.