

Only jump

Relazione del progetto per il corso di Programmazione
Università di Bologna

E. Argonni, O. Ayache, L. Peronese, S. Musiani

2022 - 2023

1 Introduzione

Only jump è un gioco platform a scorrimento verticale illimitato in grafica ASCII completamente realizzato in C++. L'esecuzione del gioco avviene interamente all'interno di un terminale, dove è possibile eseguire tutte le azioni necessarie attraverso la tastiera. Sono presenti diversi tipi di nemici e armi, queste ultime acquistabili in un apposito market tramite le monete raccolte durante la partita. Essendo centrale nel gameplay la meccanica di salto, è stata implementata quest'ultima in modo che si possa scegliere, in funzione del tempo di pressione dell'apposito tasto, la forza del salto. È stato inoltre previsto un pannello per calibrare la tastiera in modo che l'esperienza di gioco sia il più simile possibile su tutti i dispositivi in cui viene giocato.

2 Spartizione dei ruoli

Essendo il progetto sviluppato su più classi, è stato possibile lavorare parallelamente suddividendo il lavoro tra tutti i componenti del gruppo. Nello specifico la suddivisione è stata la seguente:

- *E. Argonni*: Implementazione della classe **Game**, ovvero la classe più importante di tutto il gioco in quanto si interfaccia con tutte le restanti classi. Grazie a essa infatti c'è una coesione tra tutte le porzioni di codice sviluppate separatamente. Al suo interno è possibile trovare la funzione che contiene il loop principale in cui si svolge una partita. **Menu** è la classe adibita alla gestione del menu iniziale di gioco ed è inoltre responsabile dell'animazione a inizio gioco. **Save** è classe che si occupa di interagire con l'utente quando è necessario salvare una partita. Si occupa però solo dell'aspetto grafico, mentre la classe che esegue l'effettivo salvataggio su file è differente. La **Settings** invece si occupa delle impostazioni: personalizzazione dei tasti di gioco, calibrazione della tastiera e altre impostazioni specifiche minori. Infine la classe **Statistics** che si occupa di tenere traccia della vita, delle monete e in generale delle statiche del giocatore.
- *O. Ayache*: Implementazione delle classi **Screen** e **Draw** che hanno il compito di nascondere completamente tutte le funzioni di **ncurses**¹ in due

¹**Ncurses** è la libreria esterna usata per gestire la grafica su terminale del gioco

semplici e comode classi in modo che tutte le classi avente necessità di disegnare sullo schermo potessero semplicemente interfacciarsi a loro, senza usare direttamente la libreria `ncurses`. Questo ha permesso di aggiungere un livello di astrazione maggiore che nascondesse l'implementazione a basso livello della libreria. La classe `Entity` è una generica entità nel gioco che grazie all'ereditarietà fornita dal paradigma OOP è la classe padre di `Enemy`, `Coin` e `Bullet` che da sole sono già autoesplicative. La classe `Market` si occupa di disegnare tutto il menu relativo al negozio e di fornire all'utente un metodo facile per l'acquisto di armi e abilità. Queste ultime sono rappresentate grazie a delle classi apposite: `Gun` e `Ability`. Infine la classe `Events` si occupa della gestione delle e della loro integrazione con la partita.

- *L. Peronese*: Implementazione delle mappe tramite la classe `Map` insieme alle classi `Chunk` e `Platform`. Queste ultime sono frammenti più piccoli che composti formano una mappa infinita e facilmente utilizzabile attraverso la classe apposita. Il salvataggio delle partite, delle impostazioni, come i tasti personalizzati, e la calibrazione è delegato alla classe `File`. Questa classe si occupa della effettiva scrittura su disco ed è quindi invisibile all'utente in quanto chiamata da altre classi. Inoltre si occupa di leggere i salvataggi quando il gioco viene riaperto o è necessario caricare una partita precedentemente salvata. La classe `Random` viene usata come interfaccia dalle altre classi in qualsiasi momento sia necessario generare numeri casuali per avere un luogo unico dove vengono gestiti i *seed*. Per fare debug, in quanto lo `stdout` è occupato dalla grafica del gioco, è stata scritta la libreria di funzioni `logs` che permettono di scrivere su file stringhe arbitrarie.
- *S. Musiani*: Implementazione della libreria di fisica per la simulazione di oggetti, gravità, velocità e salti. Distinta del namespace `phy` comprende le seguenti classi: `Point` per rappresentare un punto su un piano bidimensionale; `PrecisesPoint` per rappresentare un punto su un piano bidimensionale il più precisamente possibile, cioè senza approssimazioni; `Vector` per la rappresentazione di un vettore; `Body` per la simulazione di un corpo che ha una posizione, una velocità e una accelerazione. Sono inoltre presenti delle costanti nel file `constants.hpp`, come l'accelerazione della gravità in gioco e delle funzioni per simulare le collisioni tra oggetti nel file `collisions.cpp`. La scrittura della classe `Manager` che tiene traccia e gestisce tutte le entità nel gioco come monete, proiettili e nemici. Tiene traccia della loro morte, dei loro movimenti e dei loro comportamenti, anche in relazione al giocatore. Comunica inoltre con la classe per il salvataggio su file in modo da memorizzare lo stato delle entità efficientemente.

Essendo da specifiche proibito l'utilizzo della standar library di C++ e in particolare tutte le classi che facilitano l'allocazione di memoria e la gestione di essa, sono state re-implementate le classi `string` e `vector` in particolare entrambi sotto il namespace `nostd`. Esse fanno quello che ci si aspetta e sono una implementazione personale delle classi di libreria. La classe `string` è stata realizzata con *L. Peronese*.

3 Scelte implementative

3.1 Screen e Draw

Per semplificare il più possibile le operazioni di disegno su schermo e in generale tutto quello che riguardasse la grafica sono state create queste due classi per nascondere completamente la libreria `ncurses`. In particolare la classe `Screen` implementa solo le operazioni base che si possono fare su uno schermo come inizializzarlo, cancellarlo, impostarne la dimensione ecc. La classe è creata in modo che tutte le volte che un oggetto `Screen` viene creato automaticamente una nuova finestra di `ncurses` viene allocata.

La classe `Draw` estende `Screen` ereditando tutti i metodi base per gestire lo schermo, ma implementando a sua volta tutti i metodi necessari per disegnare. In particolare quelli necessari per scrivere a schermo stringhe, interi o semplici caratteri. Vengono inoltre forniti metodi per disegnare figure quali linee e rettangoli. Sono presenti anche i metodi per disegnare la mappa, il giocatore e tutti i tipi di entità quali nemici, proiettili e monete.

3.2 Game

La classe principale del videogioco è `Game`, la quale gestisce l'interazione con le altre classi e il flusso complessivo del gioco. La funzione `RUN` è la prima a essere chiamata e ha il compito di far partire tutto il gioco. In particolare esegue il menu principale del gioco che permette all'utente di scegliere tra diverse opzioni come giocare, riprendere una partita salvata, accedere alle impostazioni o uscire dal gioco. La funzione `PLAY` racchiude il ciclo principale in cui una partita viene svolta. È infatti responsabile del progresso di gioco e gestisce l'interazione tra il giocatore, le entità, le piattaforme e le statistiche. Tramite i controlli da tastiera, il giocatore può muoversi, saltare, sparare e accedere al market. La classe gestisce l'aggiornamento della posizione del giocatore e delle entità, la collisione con le piattaforme e consente di mettere in pausa il gioco per accedere alle opzioni come le impostazioni, i salvataggi e il market.

3.3 Menu

La classe `Menu` gestisce il menu principale del videogioco e contribuisce a creare un'esperienza coinvolgente sin dai primi istanti di avvio. È infatti responsabile dell'animazione di intro del gioco e della gestione del menu principale, consentendo all'utente di scegliere fra le opzioni disponibili. La funzione `GETSELECTEDOPTION` in particolare gestisce la navigazione dell'utente all'interno del menu e rende possibile la scelta delle opzioni con facilità evidenziando visivamente l'opzione selezionata.

3.4 Statistics

La classe `Statistics` fornisce la struttura per la gestione delle statistiche del gioco. Permette all'utente, durante l'esecuzione del gioco, di avere un'interfaccia in cui poter controllare in tempo reale le statistiche di gioco, il numero di monete disponibili, la quantità di vite possedute, il livello a cui si trova e il numero totale di salti effettuati.

3.5 Settings

La classe **Settings** fornisce un'interfaccia intuitiva per la gestione delle opzioni di configurazione del gioco. Consente all'utente di personalizzare la propria esperienza regolando diversi parametri come la sensibilità del salto, il volume della musica e i tasti di controllo del personaggio. Inoltre, è possibile effettuare la calibrazione della velocità della propria tastiera con cui viene ricaricata la forza di un salto. Grazie all'interfaccia grafica l'utente può navigare tra le varie opzioni utilizzando input da tastiera. Una volta che l'utente ha effettuato le modifiche desiderate ed è uscito dalle impostazioni, modifiche vengono automaticamente salvate e memorizzate per le sessioni di gioco successive.

3.6 Market, Ability, Gun e Events

Le armi, allo stesso modo delle abilità, sono rappresentate da una classe specifica, rispettivamente **Gun** e **Ability**, dove vengono differenziate per i parametri al loro interno. Per esempio il nome, il costo e il tempo di ricarica di un'arma. Questo permette di implementare una sola volta tutte le funzioni che necessitano di avere in input armi o abilità e fare i controlli necessari per distinguerle al loro interno.

Events è la classe che gestisce le abilità e il loro tempo di ricarica. Queste ultime, infatti, dopo essere acquistate rimangono al giocatore, che però le può utilizzare solo successivamente a ogni periodo di ricarica. Una barra in alto a destra nella schermata di gioco mostra se si può utilizzare l'abilità, se si sta scaricando o se si sta caricando.

3.7 Random

Fulcro di questa classe è il **seed**, ovvero un intero generato casualmente all'avvio di una nuova partita e che viene associato a essa; grazie a questo dato, la classe **Random** genera l'ordine dei chunk, la posizione delle monete e gli attributi dei nemici utilizzando la generazione di numeri casuali inclusa in C++. Questo metodo ci garantisce la assoluta unicità di ogni partita e semplifica molto il salvataggio: non è infatti necessario salvare gli infiniti valori associati all'ordine dei chunk e alla posizione delle entità, ma sarà sufficiente memorizzare il seed associato alla mappa, in quanto le funzioni pseudocasuali presenti nel linguaggio generano la stessa sequenza di numeri a partire dallo stesso seed.

3.8 Map, Chunk, Platform

Queste tre classi si occupano della gestione della mappa di gioco. La classe **Platform** definisce una singola piattaforma nel gioco, che possiede come attributi la lunghezza e le coordinate in cui si trova. La classe **Chunk** invece rappresenta una sezione del livello di gioco e come attributo possiede un vettore di oggetti **Platform**; la gestione delle collisioni si interfaccia con questa classe per conoscere se in un certo punto c'è una piattaforma. La classe **Map** infine rappresenta il livello generale del gioco; qui vengono inizializzati i 10 chunk che vengono selezionati in maniera randomica per formare l'infinita mappa di gioco; ogni mappa è caratterizzata da un **seed** univoco.

3.9 File

La classe `File` è preposta ad interfacciarsi con i salvataggi, sia dei progressi che delle impostazioni. Nel file `save.txt` vengono scritti e aggiornati tutti i valori della partita attuale, quindi nome, seed, posizione nella mappa del giocatore, entità, oggetti acquistati nel market e valori delle statistiche; invece, nel file `settings.txt` vengono memorizzati i tasti scelti dal giocatore per i comandi di gioco e le sue preferenze riguardo a volume e sensibilità. Oltre a svolgere la funzione di scrittura, la classe si occupa anche di leggere i file e trasferire i dati alla classe `Game` per ricaricare impostazioni e salvataggi memorizzati in precedenti partite.

3.10 Fisica

Tutta la libreria di fisica si raggruppa sotto lo stesso `namespace` denominato `phy`. Lo scopo di questa libreria è finalizzato soltanto alla simulazione dei salti del giocatore e al movimento dei proiettili, soprattutto quelli più complessi come quelli prodotti dal *lancia granate*. Sono quindi presenti solo gli elementi necessari a simulare il movimento di un punto nello spazio dotato di posizione, velocità e accelerazione. La posizione in particolare è rappresentata dalla classe `Point` e sia la velocità che l'accelerazione, in quanto vettori, vengono rappresentati mediante la classe `Vector`². Durante i test si è notato però che alcuni oggetti avevano necessità di memorizzare la loro posizione tramite stadi intermedi, senza quindi seguire forzatamente la rappresentazione intera della posizione fornita dalla classe `Point`. È quindi stata creata una ulteriore classe apposita identificata come `PrecisesPoint`. Non si è deciso di rimuovere completamente la precedente `Point` per una questione di ottimizzazione: fornendo una rappresentazione estremamente più precisa della posizione, la classe `PrecisesPoint` è necessariamente più pesante in memoria e più complessa da gestire. Dove quindi era possibile evitarla si è continuato a fare uso della semplice rappresentazione intera della posizione fornita dalla classe originale. La classe `Body` è una semplice rappresentazione di un corpo puntiforme privo di massa con la capacità di muoversi nello spazio: incorpora quindi tutti i campi necessari per la posizione, velocità e accelerazione.

Le collisioni tra il giocatore e le piattaforme presenti nella mappa sono gestite sempre dalla libreria di fisica. In particolare sono calcolate partendo dalla posizione attuale del giocatore, successivamente si aggiorna la sua posizione in base alla sua velocità e accelerazione. Si esegue quindi un confronto tra la vecchia posizione e la nuova, controllando che nello spostamento il giocatore non abbia colliso con nessun oggetto. Nel caso però una collisione si sia verificata, viene spostato il giocatore nel punto della collisione e vengono aggiornate velocità e accelerazione per rendere approssimativamente l'effetto del trasferimento di energia cinetica avvenuto durante la collisione, nonostante il calcolo dell'energia non sia effettivamente eseguito.

²Attenzione a non confondersi con i nomi della classi. Quella citata è più precisamente la classe `phy::Vector`, in quanto esiste anche la re-implementazione del vettore della libreria standar di C++ che invece è identificata dal `namespace nstd`