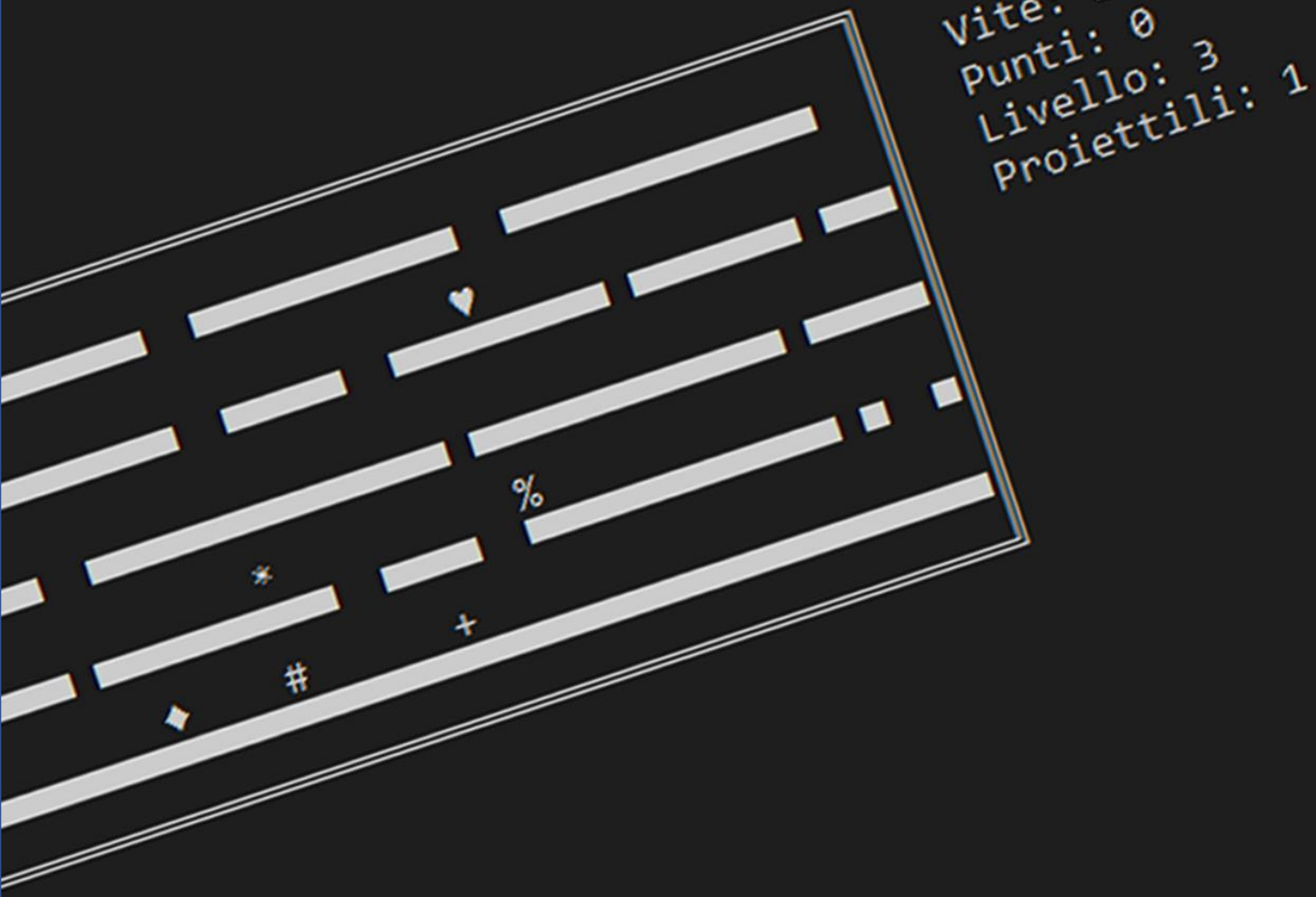

UniBros

Emanuele Di Sante
Vittorio Del Giorno

Progetto di Programmazione

C.d.S. Informatica, Università di Bologna
A.A. 2020 - 2021

Panoramica generale

IcySamurai (Vittorio Del Giorno) e LuigiBrosNin (Emanuele Di Sante) vi danno il benvenuto in 'UniBros', un semplice gioco platform sviluppato come progetto per un esame universitario!
Come in tanti altri giochi di questo genere, dovrete avanzare nei livelli generati racimolando quanti più bonus possibile ed evitando tutti i nemici che ti si pareranno davanti, senza uno scopo ben preciso.

Immaginiamo che ora vi starete sicuramente chiedendo: "Ma perché dovrei? Non c'è nessuna principessa da salvare, nessun montepremi da vincere!" ...ed effettivamente non possiamo che darvi ragione, in fondo. Però, potreste aiutarci a prendere un voto decente all'esame, ottenendo la nostra gratitudine!

Comandi di gioco

Per muoversi ed agire nel livello, serviranno solo cinque tasti:

- A (oppure ◄): sposta il tuo giocatore a sinistra;
- D (oppure ►): sposta il tuo giocatore a destra;
- W (oppure ▲): se presente, sposta il tuo giocatore sulla piattaforma soprastante;
- S (oppure ▼): se presente, sposta il tuo giocatore sulla piattaforma sottostante o sul terreno "base" di gioco;
- E: se ci sono munizioni, spara un proiettile per uccidere un nemico.

Ambiente di gioco

All’interno di ogni livello capiterà di trovare sia degli oggetti che dei nemici. Per identificarli, ecco una breve guida:

- $ - bonus punteggio: aggiunge un valore casuale di punti al punteggio attuale;
- ❤ - bonus vita: aggiunge una vita a quelle già disponibili, oltre a un esiguo valore casuale di punti;
- + - bonus munizioni: aggiunge una munizione a quelle presenti nel tuo caricatore, oltre a un esiguo valore casuale di punti;
- ▲ - nemico statico: è il nemico più semplice da affrontare, proprio perché non effettua alcuna azione;
- % - nemico dinamico: è il nemico di livello intermedio, capace di muoversi in una delle quattro direzioni (scelta casualmente quando viene generato);
- # - nemico sputafuoco: è il nemico più difficile da affrontare, in quanto spara delle palle di fuoco a ripetizione in una delle quattro direzioni (scelta casualmente quando viene generato);

Per raccogliere un bonus basterà posizionarsi sulla sua icona; per eliminare un nemico si può scegliere se spararegli o calpestarlo. Ognuno di essi potrà rimuovere una vita se gli si passa attraverso - anche se così facendo lo si eliminerà - e il nemico sputafuoco sparerà palle infuocate senza fermarsi mai, a meno che non lo si elimini. Inoltre, è possibile sia avanzare di livello che retrocedere: basterà arrivare ai limiti orizzontali dello spazio di gioco per cambiare livello, trovando tutto quello che hai lasciato quando hai cambiato area.

Struttura del progetto

Abbiamo deciso di creare una soluzione C++ su Visual Studio, soprattutto per effettuare comodamente le operazioni di debug: il compilatore integrato permette di visualizzare le celle di memoria relative ad ogni oggetto / struttura / variabile istanziato in un dato momento interrompendo l'esecuzione del programma.

Sfruttando la programmazione ad oggetti abbiamo creato diverse classi (figura 1), delle quali però le principali sono:

- **Unibros**, il '**main()**' del progetto;
- **Entity**, la classe padre di tutte le entità presenti nel gioco;
- **Screen**, che si occupa di gestire la generazione e la visualizzazione dello spazio di gioco.

L'approccio scelto ci ha permesso di sfruttare l'ereditarietà delle classi, impostando '**Entity**' come classe padre di tutte le altre (eccetto quelle sopra menzionate): gli oggetti generati nei livelli, infatti, sono tutti uguali strutturalmente parlando, perché devono essere gestiti tutti allo stesso modo dal codice. Ciò che differenzia un oggetto '**Bonus**' da un oggetto '**Enemy**', infatti, è solo l'effetto che ha sul giocatore - l'oggetto '**Player**' creato all'avvio del gioco - ma praticamente devono essere tutti posizionati nel livello allo stesso modo, cioè sfruttando una specie di "cursore invisibile" che va a stampare nella posizione desiderata l'oggetto in questione (vedi funzioni '**MoveCursor()**' e '**PrintAt()**' in '**Funzioni**').
La classe '**Screen**', invece, si occupa di creare diverse strutture dati contenenti array di *i* nemici, *j* bonus e *k* piattaforme, con *i* *j* e *k* scelti tenendo conto della difficoltà *l* del livello, che cresce man mano che il giocatore avanza nel gioco. All'interno di questa struttura è presente un puntatore al livello successivo e uno al precedente, creando quindi una "doppia lista" che permette di navigare tra i livelli generati.

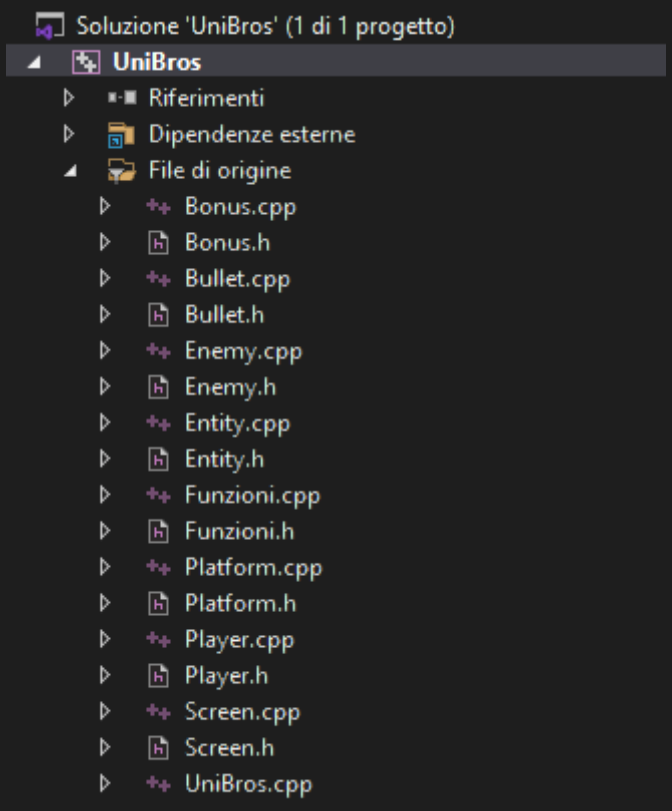

Figura 1 - I file .cpp e .h del progetto

Contenuto del "main()"

Nel momento in cui viene lanciato il gioco si entra in un ciclo while(), dal quale si uscirà solo se le vite del giocatore termineranno o se si sceglierà di terminare volontariamente la partita. In questo ciclo vengono chiamate a ripetizione le seguenti funzioni e/o metodi (figura 2):

- **Movement()**, presente in '**Funzioni**', che si occupa di gestire le azioni del giocatore quando viene premuto un tasto dall'utente;
- **Handler()**, presente in '**Funzioni**', che garantisce il rispetto dei limiti di gioco da parte del giocatore e la legittimità dei suoi attributi (vite, munizioni e punteggio);
- **Print()**, presente in '**Screen**', usato per stampare i contenuti dello schermo;
- **Print()**, presente in '**Entity**', usato per stampare a una coordinata (x, y) il giocatore.

```
int main()
{
    Cursore(false);
    WaitScreen();

    while (!gameOver)
    {
        // Creazione del gestore per gli input da tastiera
        Movement(&gameOver, p, schermo);
        Handler(width, height, &gameOver, p, schermo);

        // Stampa a schermo di tutte le componenti del livello
        schermo.print();
        p.print();
        PrintInfo(width, height, p, schermo);

        Sleep(50);
    }

    Clear();
    GameOverScreen();
}
```

Figura 2 - il main()

Gli unici dati salvati sono gli oggetti contenenti i dati del giocatore e dello schermo, che vengono istanziati nel **main()** e dichiarati variabili globali: *p* per il primo, *schermo* per il secondo. Come già accennato, ogni qualvolta "**Movement()**" rileva uno spostamento che "**Handler()**" traduce in un cambio di livello viene chiamato il metodo "**nextLevel()**" di '**Screen**', e generato un nuovo livello tramite il metodo "**generateLevel()**" della stessa classe, oppure viene chiamato il metodo "**prevLevel()**" che tramite il puntatore al precedente livello permette al giocatore di tornare indietro.

Il metodo "generateLevel()"

Tutti gli oggetti del livello vengono generati in "**generateLevel()**": a seconda del parametro intero "**difficoltà**" i metodi "**generatePlatform()**" di '**Platform**', "**generateBonus()**" di '**Bonus**' e "**generateEnemies()**" di '**Enemies**' creano la matrice di piattaforme, la lista di bonus e la lista di nemici. La prima viene generata tramite un ciclo for() che scorre tutti gli elementi: per ogni coordinata viene scelto se generare o meno una piattaforma, usando la funzione "**srand()**" e la difficoltà in quel momento; sulla base (letteralmente) delle piattaforme appena create, i bonus vengono generati sfruttando nuovamente "**srand()**" per le coordinate e il valore della difficoltà attuale per il loro numero, mentre il tipo affibbiatogli è

del tutto casuale; i nemici, invece, sfruttano la difficoltà per decidere in quanti essere e la funzione "**srand()**" per decidere il tipo, la posizione ed eventualmente la direzione in cui dirigersi.
Sia nel caso dei nemici che dei bonus, la posizione in cui essere generati viene scelta verificando che ci sia una piattaforma al di sotto di essi, sfruttando il metodo "**isThere()**" di '**Platform**'.

Gestione della partita

Ogni volta che viene preso un bonus si otterrà un incremento del punteggio di un valore casuale; inoltre, nel caso del "bonus vita" si otterrà una vita in più, mentre nel caso del "bonus proiettile" si otterrà un proiettile in più.

I nemici, come già accennato, possono essere statici o dinamici. Nel secondo caso, possono muoversi secondo un unico asse: al momento della generazione di un nemico viene inizializzato l'attributo "**direction**" che specifica verso quale direzione esso si debba muovere; nel caso in cui non sia possibile muoversi verso tale direzione (generazione ai limiti dello schermo o ad un bordo di una piattaforma), il nemico si muoverà nel verso opposto. Nel caso in cui un nemico è in grado di muoversi sull'asse delle ordinate, sarà in grado di volare e quindi di spostarsi anche se non trova una piattaforma sotto di lui, a differenza di quelli che si muovono sull'asse delle ascisse.
Lo stesso tipo di controllo riguardante i limiti di gioco è applicato anche per i nemici statici in grado di sparare (i nemici sputafuoco): dato che sarebbe inutile sparare contro un bordo di schermo, se la generazione avviene in un posto troppo conveniente al giocatore la direzione del proiettile viene invertita. La dinamicità di questi nemici è data dal fatto che il metodo "**print()**" di '**Screen**' viene eseguito costantemente, anche se si rimane nello stesso livello. Infatti, è proprio per questo che si potranno vedere sia i movimenti del giocatore che quelli effettuati dai nemici.

Per eliminare un nemico, come già detto, l'utente sceglierà se saltargli sopra o far sparare al proprio giocatore un proiettile, se possibile. Il metodo che si occupa della visualizzazione dell'oggetto '**Bullet**' a schermo è "**fireb()**", che gli assegna le coordinate del giocatore; successivamente, il proiettile in questione viene collegato logicamente allo schermo tramite il metodo "**setBullet()**" e poi gli verrà incrementata la coordinata x ad ogni aggiornamento dello schermo, fino a che non scomparirà dal livello.

La schermata "**Game Over!**" verrà infine visualizzata quando il giocatore terminerà le vite a disposizione, oppure se viene premuto il tasto "X" dall'utente.

Buon divertimento!