
**Andrea Napoli**
**0000988997**

**Manuel Arto**
**0000974775**

***Progetto di Programmazione***

# Platform Game

Link Github

**Anno accademico 2020/2021**

# 1. Introduzione e descrizione del gioco

---

Il nostro progetto è un platform game, ovvero un gioco costituito da piattaforme di cui il giocatore si serve per muoversi all'interno di esso, per schivare nemici o per raccogliere bonus. Man mano che il giocatore avanza tra i livelli, i nemici acquisiscono sempre più attacco e più vita.
Il gioco si conclude quando la vita arriva a 0 e verrà visualizzato a schermo il punteggio ottenuto.

L'obiettivo del giocatore è quello di totalizzare più punti possibili e ciò avviene o attraverso il raccoglimento di alcuni bonus o eliminando i vari nemici sparando. Inoltre ogni nemico ha delle diverse caratteristiche (attacco, vita, ecc..) che variano in base al livello raggiunto. Ne troveremo 3:

- **EasyEnemy**: un missile che viene generato alla stessa altezza del giocatore e dal fondo dello schermo
- **MediumEnemy**: un nemico che viene generato su una piattaforma e che si muove avanti e indietro su di essa
- **HardEnemy**: un nemico che oltre a sparare ha la capacità di seguire il giocatore

Vi sono anche 4 tipi di bonus:

- **Invincibility**: ti rende invincibile per un tempo limitato
- **Points**: aumenta il punteggio
- **Life**: aumenta la vita
- **Minigun**: permette di sparare più veloce per un tempo limitato

Inoltre all'aumentare del livello, il gioco diventa più difficile: la vita e l'attacco dei nemici aumenterà, si muoveranno e spareranno più veloce, ma anche i bonus diventeranno più fruttuosi.

# 2. Descrizione del progetto e implementazione

---

Abbiamo implementato il nostro progetto attraverso il pattern **MVC** (Model-View-Controller) che ci permette di separare i ruoli tra le varie classi.
Ogni classe è suddivisa in due file:

- *.hpp: header in cui vengono definite le variabili e i metodi della classe
- *.cpp: vengono implementati i metodi

Tabella classi:

| CLASSE | DESCRIZIONE |
|---|---|
| Controller | Nel Controller viene gestito tutto il flow del programma, ha accesso alla **View**, alla classe **Generator** e gestisce lo stato del **Player.**<br>Si occupa di inizializzare il programma e effettuare i vari controlli.<br>Tutto il flow dell'esecuzione è controllato nel metodo run() della classe in un ciclo do-while |
| Generator | Il Model del nostro pattern, si occupa di gestire tutti i dati del programma come la gestione delle stanze e la loro generazione pseudo-random, la gestione dei vari nemici e bonus attraverso liste dinamiche |
| View | Ha accesso alla libreria **ncurses**, si occupa di tutto ciò che l'utente visualizza sullo schermo |
| Character | Classe padre da cui ereditano la classe **Player** e le 3 classi dei nemici.<br>Si occupa di gestire le variabili in comune a tutte le entità e implementa dei metodi riguardanti il movimento e lo sparo (implementato tramite struct e lista dinamica). |
| Collisions | Si occupa della gestione di tutte le possibili collisioni che possono avvenire nel gioco, ovvero:<br>- collisione fisica tra giocatore e nemici,<br>- collisione tra spari dei nemici e giocatore e viceversa<br>- collisione tra gli spari e piattaforme o muri<br>- collisione tra giocatore e bonus |

| | |
|---|---|
| Platform | Implementazione di una piattaforma tramite classe attraverso l'utilizzo di 3 variabili: x, y, length.<br>Implementa inoltre dei metodi statici per il controllo di piattaforme sopra, sotto, a sinistra o a destra data una coordinata |
| Player | Eredita da **Character** e si occupa di tutto quello che riguarda il giocatore, ovvero il movimento, l'incremento della vita e dei punti, il timer dei bonus e l'attivazione di essi. |
| EasyEnemy | Eredita da **Character** e implementa un metodo move per la gestione del movimento |
| MediumEnemy | Eredita da **Character** e implementa un metodo move che gestisce un movimento avanti e indietro sulla piattaforma di spawning.<br>Implementa inoltre un override del metodo shoots per far sì che il nemico spari da una distanza minima |
| HardEnemy | Eredita da **Character** e implementa un metodo follow che permette al nemico di seguire il giocatore.<br>Implementa inoltre un override del metodo shoots per far sì che il nemico spari da una distanza minima |
| Bonus | Implementazione tramite classe di un bonus attraverso l'utilizzo di variabili per le coordinate e per la gestione dei 4 tipi diversi |

**Principali** scelte nell'implementazione del progetto:

- Gestione del movimento del giocatore:

  Il giocatore ha la possibilità di muoversi a destra o a sinistra di 1 e la possibilità di saltare di 2.

  Nel caso in cui il movimento a destra o a sinistra avvenga in aria, il player si sposta a destra/sinistra di 2 (così da simulare un salto in diagonale)
- Generazione delle stanze pseudo-random:

  Abbiamo deciso di implementare le stanze utilizzando un array che salva contemporaneamente due stanze adiacenti (la stanza di sinistra nella prima metà dell'array e la stanza di destra nella seconda metà).
  Ogni volta che ci muoviamo in una stanza non salvata nell'array: viene rimossa la stanza più lontana, si effettua lo shifting di quella più vicina nell'altra parte dell'array e infine viene generata la nuova stanza attraverso una funzione di PRNG che tramite i due input, ovvero seed

(inizializzato a time(0) a ogni istanza del programma) e room, genera sempre lo stesso numero pseudo-random che viene successivamente mappato da 0 a 9 per l'ottenimento di 1 dei 10 template da noi creati

- Gestione movimento dinamico nelle stanze:

  Abbiamo deciso di gestire l'avanzamento nel gioco mantendo fissa la posizione del player e facendo scorrere la mappa dietro di lui.
  Per fare ciò nella classe Player sono state aggiunte due variabili:

  FIXED_X: massima coordinata x che il player può raggiungere
  offset: rappresenta la differenza rispetto al valore fisso

  All'avvio del gioco il player si troverà a coordinata x=0 e muovendosi a destra, dopo aver raggiunto la sua posizione fissa (FIXED_X), inizierà ad aumentare invece l'offset.
  Ogni entità che incontreremo nel gioco sarà quindi creata tenendo conto dell'offset a cui ci troviamo.
  Inoltre per il controllo con le altre entità nel gioco abbiamo effettuato l'override del metodo getX() nella classe Player, che ritornerà la coordinata x del player sommata all'offset.
  Mentre per quanto riguarda la stampa degli elementi nello schermo, ogni volta che una entità deve essere stampata viene prima controllato che la sua coordinata x meno l'offset risulti compresa tra 0 e la dimensione della schermata di gioco

- Spawning di bonus e nemici:

  Ogni nemico o bonus viene generato sopra una piattaforma scelta random all'interno della stanza in cui ci troviamo.
  Il numero di bonus per stanza può variare da 0 a 3 con diverse probabilità per ognuno.

# 3. Suddivisione dei compiti

---

**NAPOLI**:

- Creazione dell'interfaccia grafica (mappa, scritte, posizioni nello schermo, …)
- Gestione dell'ereditarietà
- Gestione del movimento e dello sparo dei nemici
- Gestione delle piattaforme e controlli per giocatore, nemici e spari
- Gestione delle varie collisioni
- Creazione dei template per ogni stanza
- Definizione del gioco con i vari punteggi e regole

**ARTO**:

- Implementazione dell'interfaccia grafica
- Implementazione degli spari tramite lista dinamica
- Gestione dei nemici e implementazione tramite liste dinamiche
- Gestione del movimento del giocatore
- Gestione dei bonus e implementazione tramite lista dinamica
- Gestione movimento dinamico nelle stanze
- Generazione dinamica pseudo-random dei template
- Implementazione di random-spawning di nemici e bonus