Progetto di Programmazione A.A. 2021/2022

Federico Turrini 0001030984
Manuel Flagelli 0001031994
Mattia Malservigi 0001029225
Ahmed Dhahri 0001019808

10 gennaio 2023

1 Introduzione

In questa breve relazione si esporranno come richiesto le principali scelte progettuali, l'implementazione delle funzioni salienti, e la suddivisione del lavoro tra i membri

1.1 Gioco

Il gioco come da requisiti non presenta traguardi particolari, il giocatore deve uccidere tutti i nemici in una stanza per poter prendere la chiave che sblocca i portali. Una volta ottenuta la chiave di una stanza, sarà possibile accedere ad altri punti della mappa che potrebbero anche nascondere degli artefatti. Una volta completate tutte le stanze il gioco è sostanzialmente finito. Alcuni nemici sono più forti di altri e avranno bisogno di più colpi per essere eliminati.

1.2 Comandi

Il giocatore muove il personaggio con i tasti WASD (W: in alto, A: a sinistra, D: a destra, S: in basso), premendo il tasto "shift" + WASD si fa un salto e infine spara con il pad direzionale.

2 Struttura del Gioco

2.1 Mappa

2.1.1 Introduzione

La mappa del gioco è strutturata come un albero binario, quindi ogni stanza rappresenta un nodo del suddetto. Il Personaggio Giocante (da qui in poi chiamato PG), è "libero" di esplorare la mappa nella sua interezza attraversando dei portali.

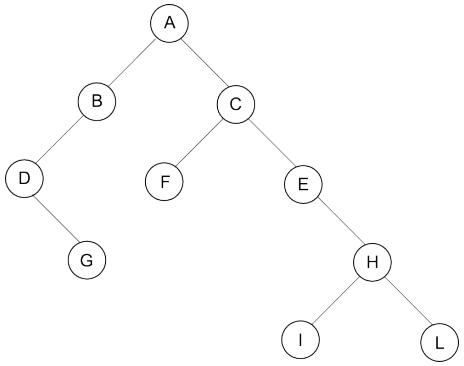

Figura 1: Una possibile configurazione della mappa con 10 stanze

2.1.2 Stanze

Le stanze vengono generate in modo randomico tra 4 possibili layout, questi si trovano in "draw.hpp". Ogni qual volta che il PG attraversa un portale, viene generato un nodo nell' albero o a destra o a sinistra. Altrimenti se il PG aveva precedentemente visitato quella stanza, essa viene semplicemente ridisegnata grazie ai dati già presenti nella struttura dati. Si è deciso di procedere in questa direzione per rendere l'esperienza di gioco il meno monotona possibile.

```
struct mappaAlbero
{
    Room val = Room(acroom,acsizet,aclayout,acporta,acId);
    mappaAlbero* left;
    mappaAlbero* right;
    mappaAlbero* parent;
    int enemies_life[3];  // vita nemici
    bool already_added[3]; // memorizzazione dello score
    bool nohpup;   // salvataggio stato artefatto
    bool key_taken;  // se la chiave è gia stata presa
    bool key_used_right;  // chiave usata porta destra
    bool key_used_left;  // chiave usata porta sinistra
    bool delone;  // se sono state raccolte delle monete
    bool nocoins;  // se ho raccolto tutte le monete
    bool first_map;  // controllo se sono nella prima mappa
    lartifact monete_attive = new artifact_list;
};
```

Figura 2: Struttura dati usata per l'implementazione della mappa

2.2 Nemici e PG

2.2.1 Introduzione

Il PG e i nemici sono le uniche entità dotate di movimento, il PG viene manovrato dal giocatore, mentre i nemici errano per la stanza seguendo un pattern pre-stabilito. Entrambe le entità sparano dei proiettili che si distruggono a contatto coi muri.

2.2.2 SpawnPoint

Il PG viene generato nella mappa in punti fissati, a seconda della conformazione del layout. I nemici vengono generati casualmente in 3 possibili punti diversi per ogni layout, diminuendo ulteriormente la monotonia del gioco

2.2.3 Movimento e Generazione Proiettili

Il movimento dei nemici, del PG e dei proiettili sono basati sui frame. Ad ogni ciclo del gioco vengono aggiornate le coordinate di ogni entità, uno dopo l'altro secondo la loro posizione dentro la mappa e secondo l'input del giocatore. I proiettili sono implementati con 2 liste, una che crea i proiettili ogni volta che vengo sparati dal PG e un altra che li elimina quando non servono più. Uno switch gestisce gli input del PG, esso presenta anche una funzione di schivata che lo fa saltare di due caselle in ogni direzione; ha un cooldown gestito da una variabile che blocca il getch() per un determinato tempo.

2.3 Collisioni e Creazione Entità

L'interazione e le collisioni tra il PG, gli artefatti ed i nemici è dettata dalla classe playerEnemy.hpp. Qui si sviluppa il ciclo del gioco per ogni singolo layout. I nemici e il PG vengono creati con i valori passati come parametri dall'albero, quindi si tiene traccia di tutti gli aggiornamenti e degli eventi che avvengono durante il gioco. Tramite la funzione mvwinch(), che rileva il carattere in una certa posizione, si gestiscono le collisioni e l'eventuale raccolta di un artefatto.

```
if(!nemico3->return_kill())
{
    nemico3->leftenemsh();
}
else if(nemico3->currentlife()<=0 && !already_added[2])
{
    nemico3->undisplaybullets();
    score = score+50;
    already_added[2]=true;
}
if(!hpup->return_remove()){        //serve a raccogliere l'artefatto
    player->healtup(hpup, score);
    if(hpup->return_remove()){      //costa 100 score raccoglierlo, se no non lo prende
        life = life+50;
        score = score-100;
    }
}
if(mvwinch(playwin, player->currentY(), player->currentX())=='1' ||
mvwinch(playwin, player->currentY(), player->currentX())=='2' ||
mvwinch(playwin, player->currentY(), player->currentX())=='3')
{
    life = life-50;     //collisioni player
}
```

Figura 3: Estratto di codice che gestisce collisioni e artefatti

2.4 Artefatti

2.4.1 Introduzione

Gli artefatti che possono essere presenti in una stanza sono: monete, chiavi e pozioni. Quando il PG raccoglie passandoci sopra uno di questi oggetti, si triggerano determinati eventi: le monete incrementano lo score, le pozioni riprisitinano la vita e le chiavi aprono i portali.

2.4.2 Pozioni e Monete

La classe artefatto implementa le pozioni, queste hanno un costo di 200 score per essere raccolte, le sottoclassi implementano le chiavi e le monete. Come sopra citato le collisioni e l'interazione con gli oggetti sono gestite dalla classe playerEnemy.hpp, quindi in questa classe saranno presenti le funzioni per la gestione di questi elementi. Le monete sono implementate tramite una lista, hanno diverse configuarazioni di spawn che variano a seconda del layout della mappa in cui ci si trova.

2.4.3 Chiave

La chiave, appunto, permette di sbloccare i portali che permettono la navigazione della mappa. vengono gestite dala classe "key", che è una sottoclasse di "artifact" e gestisce gli attributi legati alla chiave all'interno del gioco: a ogni stanza è associata una chiave univoca, la quale attribuisce al giocatore la capacità di aprire i portali di quella stanza. Per ottenerla è necessario uccidere tutti i nemici e, dopo averla raccolta da terra, è necessario avvicinarsi al portale per aprirlo.

3 Suddivisione del Lavoro

3.1 Suddivisone del Lavoro

Il lavoro è stato suddiviso, per quanto riguarda le classi e/o funzioni più rilevanti, nel seguente modo:

- Ahmed Dhahri ha lavorato maggiormente sul movimento delle entità e sulla meccanica dei proiettili.
- Federico Turrini ha lavorato maggiormente sulla mappa, sulla gestione delle stanze e sui layout
- Manuel Flagelli ha lavorato maggiormente sulla gestione delle chiavi , delle porte e sulla gestione del PG
- Mattia Malservigi ha lavorato maggiormente sugli artefatti, sulla gestione delle collisioni e sul menù di avvio

Nella correzione di bachi e nelle parti software meno significativi tutti i membri hanno partecipato in egual misura.