ANNO 2022-2023

@'S ADVENTURES

PROGETTO PROGRAMMAZIONE

UNIBO

MEMBRI GRUPPO
EMILIO GIACOMONI
FRANCESCO BIGLIERI
DAVID DINU

Progetto di Programmazione Anno 2022/2023

@'s Adventures

INDICE

Componenti del gruppo

Emilio Giacomoni:

Gestione delle classi player, enemy, proiettili e del sistema di movimento e cancellazione delle entità .

mail universitaria: emilio.giacomoni@studio.unibo.it

Matricola: 0001069281

David Dinu:

Gestione e creazione mappe e shops, gestione e generazione stanze, gestione oggetti/potenziamenti e salvataggio su file.

Mail universitaria: david.dinu@studio.unibo.it

Matricola: 0001090449

Francesco Biglieri:

Gestione classe game, input e layout del gioco, scelte grafiche e implementazione su main.

Mail universitaria: francesco.biglieri@studio.unibo.it

Matricola: 0001090725

1. Introduzione al Gioco

Il progetto è un gioco platform in cui il giocatore '@' si muove all' interno di stanze con piattaforme generate in maniera random a partire da diversi template. Il giocatore può saltare sulle piattaforme per schivare i proiettili nemici, colpire meglio i nemici sopraelevati e comprare oggetti nei livelli mercato presenti ogni cinque livelli. I potenziamenti acquisiti rimangono salvati tra una partita e l'altra a patto che si clicchi 'continua la partita'.
Il gioco termina quando la vita del personaggio arriva a zero oppure si preme 'e'.
L'obiettivo del gioco: è percorrere il maggior numero di stanze possibile senza morire, uccidendo i nemici e guadagnando valuta per comprare i potenziamenti

2. CLASSI DEL GIOCO

2.1. GAME

La classe **Game** è la vera e propria base del gioco perché senza di essa sarebbe difficile gestire le varie parti del gioco. **Game** prende i puntatori di tutte le classi per poter utilizzare qualsiasi funzione durante la partita, viene utilizzata per gestire principalmente questi update:

- Il salto del giocatore tramite un ciclo for.
- Il movimento dei nemici.
- Il danno ricevuto e i proiettili avversari.
- Il controllo del negozio.
- Gestione del menù (utilizzando il valore preso da **layout->main_menu()**).
- Aggiornamento degli acquisti fatti nel negozio.
- Gestione dei proiettili: movimento, danno.
- Movimento del personaggio giocabile.
- Gestione proiettili giocatore.

la funzione **RUN** serve per non chiudere il gioco in attesa di una **scelta** nel menù principale: Eseguendo **gameLOOP** se il giocatore avvia una partita. Questa funzione utilizza un ciclo while che controlla se **quit** (valore booleano che viene impostato a vero quando si preme 'e' nel gioco) non è vero, al momento del **GAME OVER** , il giocatore viene rimandato al menù principale, salvando i vari potenziamenti ottenuti nella scorsa partita in caso si dovesse selezionare **CARICA PARTITA**. L'opzione **ESCI DAL GIOCO** fa terminare l'esecuzione.

2.2. LAYOUT

La classe Layout gestisce ogni aspetto di visualizzazione del gioco, è la classe che ha accesso diretto a **NCURSES** che permette di visualizzare lo schermo, gestire la finestra del gioco e delle statistiche rendendole modificabili e visualizzabili dalle altre classi.

2.3. PLAYER

La classe **Player** gestisce un personaggio di gioco con parametri come posizione, salute, scudo, una valuta, capacità di salto e vita. Interagisce con una mappa di

gioco e ha metodi per visualizzazione, inizializzazione, movimento, aggiornamento delle statistiche, impostazione dello scudo e rigenerazione della salute.

2.4. PROIETTILE

La classe **Proiettile** definisce le proprietà e le azioni dei proiettili. È stata anche creata una struttura `***lista_proiettili***` per una lista collegata di proiettili, e funzioni per inserire, muovere, disegnare e eliminare proiettili in base a collisioni, posizione, e interazioni con il giocatore e la mappa.

2.5. ENEMY

Definisce la classe **Base_en** per i nemici con attributi come posizione, salute, e simbolo, e metodi per disegno, movimento, e sparare proiettili. Utilizza liste collegate per mantenere e aggiornare dinamicamente i nemici e i proiettili in gioco, con funzioni per aggiungere nemici alla lista, rimuovere quelli colpiti, e gestire la logica di collisione. La logica di interazione tra giocatore, nemici, e proiettili è centrata su movimenti, attacchi e la gestione dinamica della memoria. Inoltre una volta eliminato un nemico la valuta del giocatore aumenta. Vengono anche gestite le collisioni tra diverse entità ed il contatto con il player.

2.6. OBJECTS

La classe **objects** gestisce il tipo, il prezzo e la quantità di oggetti e/o potenziamenti presenti e posseduti dal giocatore. La classe ha un funzionamento molto semplice, essa è composta da un array di 9 elementi (pari al numero di potenziamenti esistenti) di tipo **object**.
Objects non è altro che una struttura che racchiude al suo interno nome, descrizione, prezzo, simbolo (identificativo) e n° posseduto, quest'ultimo è anche l'unico valore che verrà modificato durante le partite. Il costruttore assegna direttamente quelli che sono i valori standard di ogni oggetto, mentre invece saranno funzioni come **buy_object** o **load_possesion** a modificare le variabili. Sono poi presenti diverse funzioni **get** per permettere di accedere ad alcuni dati più facilmente.

2.7. PLATFORM

La classe platform è la classe che gestisce la creazione e (in parte) la posizione delle piattaforme del gioco, è caratterizzata da valori quali X, Y e lunghezza della piattaforma più un ulteriore valore che indica il numero di piattaforme (per ogni determinato template) .
Possiede poi due funzioni **platformType** e **shops_type** che generano quelli che sono i template, il primo genera un piccolo numero di piattaforme che verranno usate

per la costruzione dei livelli normali, il secondo invece genera il layout di uno dei 5 possibili shop.

2.8. MAP

La classe **Map** si appoggia alla classe platform e si occupa di generare quelle che saranno le mappe vere e proprie. E' caratterizzato da due array di piattaforme (uno per le singole piattaforme e uno che rappresenta lo shop), un booleano che rappresenta se lo shop è stato usato, ed infine una matrice di booleani la quale è true se nel caso in cui in tale posizione è presente una piattaforma. La classe usa funzioni differenti per la creazione della prima mappa, delle mappe normali e degli shop (maggiori dettagli nella sezione implementativa) oltre che per il controllo della posizione di determinate piattaforme.

2.9. ROOM

La classe **Room** si occupa delle generazione e gestione delle stanze, sia per la generazione delle mappe che per quella dei nemici. E' caratterizzato da due valori *last_room* e *current_room* (che indicano rispettivamente l'ultima stanza generata e la stanza in cui si trova il player al momento) e due array dinamici *normal_maps* (array di puntatori a oggetti map) e *room_enemy* (array di puntatori a liste di oggetti enemy). E' poi dotata di funzioni quali la generazione di nuove mappe e nemici (con la rispettiva modifica dell'array) e il *salvataggio* di mappe e nemici già creati in passato.

3. Scelte Implementative

3.1. GESTIONE DELLE ENTITÀ

Il giocatore può muoversi a destra ed a sinistra con le frecce e i comandi (←, →, A, D), può saltare in verticale con (↑, W) e, mentre è in salto, muoversi a destra e sinistra con lo stesso metodo. Inoltre può sparare premendo spazio fino a cinque volte di fila per poi ricaricare, anche saltando. I proiettili sono gestiti tramite una lista di puntatori ad oggetti **proiettile** che si cancellano una volta che escono dallo schermo, colpiscono un nemico o una piattaforma.

I nemici sono divisi in tre tipi dal costruttore della classe **enemy** (X, Y, Z), vengono distribuiti in maniera **semi-casuale**, non troppo vicini al giocatore, nella corrente stanza di gioco al momento della sua generazione e variano di simbolo e statistiche. I nemici 'X' e 'Z' si spostano verso il giocatore mentre i nemici 'Y' non si muovono. Gli 'X' non sparano mentre gli 'Y' e gli 'Z' sparano dopo un numero prefissato di cicli.

Tutti i tipi, se si trovano direttamente alla destra o alla sinistra del giocatore, gli infliggono danni da contatto. I proiettili dei nemici vengono gestiti con una funzione di cancellazione diversa che impedisce ai nemici di farsi danni a vicenda. Per ogni nemico viene creata una lista di puntatori ad oggetti proiettile. I Nemici di ogni stanza sono gestiti tramite un array dinamico di liste di puntatori ad oggetti della classe **Base_en**. All'aumentare del numero di stanza i nemici aumentano gradualmente (+1 'X' ogni stanza, +1 ' Y' ogni tre 'X', +1 'Z' ogni tre 'Y').

3.2. GESTIONE DELLA MAPPA

La mappa viene gestita sotto forma di un array di booleani il quale è true in prossimità di una piattaforma (pavimento incluso). Per generare quindi la mappa, si utilizza la funzione **platformType** (presente nella classe platform) la quale restituisce una tipologia di piattaforma casuale tra 10 template diversi che verrà poi posizionata in una posizione casuale dello schermo (che viene diviso in una griglia 4 x 3 e numerato da 1 a 12) tramite la funzione **cell** in modo tale da non avere mai una una tipologia di piattaforma nelle dirette vicinanze, ma che allo stesso tempo sia possibile raggiungerla dalla piattaforma creata in precedenza.
E' però possibile avere più tipologie di piattaforme sulla stessa riga (con una probabilità del 30% per ogni tipologia di piattaforma generata) ma queste non saranno **mai adiacenti**. In sostanza l'utilizzo casuale di 1 tipo di piattaforma tra 10 diversi template posizionato in una posizione casuale tra 12 possibili (contando anche la possibilità di alcune piattaforme di generarsi in modo casuale) permette una vasta varietà di mappe generabili.

3.3. GESTIONE DEI NEGOZI

La generazione dei negozi è simile a quella delle mappe, ma con una minore presenza di randomicità, si tratta infatti di una serie di 5 template dal quale ne viene scelto uno casuale alla generazione di uno shop.
Una però sostanziale differenza rispetto ai livelli normali è la presenza di 'oggetti' acquistabili e presenti sotto forma di scatole (**| # |**), il posizionamento degli oggetti è però arbitrale, infatti questi vengono posizionati in concomitanza di una serie di piattaforme posizionate in modo specifico (una piattaforma **lunga 5** posizionata al di sopra di una piattaforma **lunga 9**, essendo quindi i livelli dei template fatti antecedentemente è quindi facile decidere in precedenza la posizione degli oggetti.
Il tipo di oggetto/box posizionato è invece **casuale** ed una volta acquistato un oggetto, tutti gli oggetti nello shop scompaiono. Per attuare questa operazione (essendo che viene costantemente eseguito un erase e una riscrittura dello schermo) è stato implementato un valore bool relativo alla stanza il quale indica l'acquisto o meno di un oggetto e che impedisce direttamente la stampa a schermo degli oggetti in caso di acquisto. Invece per permettere che gli oggetti vengano scelti casualmente ma in modo tale che ad ogni ristampa rimangano identici, si va a fare un'**operazione sul seed** della funzione rand, in particolare si lega il seed a **4 valori**

distinti, un valore creato all'*avvio del programma*, un valore legato alla *stanza* e due valori legati alla *posizione dell'oggetto*, in questo modo si evitano eventuali uguaglianze tra oggetti (anche in caso di layout identici, ma in stanze o partite diverse).
Il personaggio è in grado di acquistare oggetti solo nel caso in cui si trovi **davanti** ad uno di loro e disponga di sufficiente monete, per riconoscere un oggetto viene costantemente controllata la stringa 3 posizioni prima e dopo il personaggio e se questa corrisponde ad una delle stringhe identificative di un oggetto appariranno le informazioni dello stesso e sarà possibile acquistarlo (in questo modo aumenterà anche il numero di oggetti in posseso).

3.4. GESTIONE DELLE STANZE

Le stanze vengono gestite dalla classe room e si basano sull'utilizzo delle classi map e enemy, vengono infatti generati due array dinamici di mappe e nemici in modo tale che gli indici rappresentino direttamente il numero della stanza corrispondente.
E' quindi possibile grazie ai valori current room e last room decidere se creare una nuova stanza o caricare una delle stanze presenti nell'array, nel pratico, se vogliamo andare in una stanza (i) si ritornano gli elementi **normal_map[i]** e **room_enemy[i]** a meno che **i** non sia uguale a **last_room**, in quel caso si chiamano le funzioni **generate_new_room** e **generate_enemy** i quali per prima cosa allungheranno l'array (creandone uno ex novo e riempiendolo a dovere) ed genereranno in ultima posizione una nuova mappa e una nuova lista di nemici.
La prima stanza verrà senza generata dalla funzione di generate_first_map (classe map) mentre invece quelle successive vengono generate casualmente dalla funzione **generateMap** mentre i nemici sono generati dalla funzione **generate_enemy** (per il numero di nemici generati controllare la gestione entità 3.1.). Ogni 5 stanze le funzioni **generateMap** e **generate_enemy** vengono sostituite dalla funzione **generate_shop** che genererà uno shop casuale tra i template disponibili.

3.5. GESTIONE DEL SISTEMA DI SALVATAGGIO

Il gioco dispone di un sistema di salvataggio su file che fa uso di due funzioni molto semplici, **saveGame** e **loadGame**. La funzione save game si occupa di salvare alcune basilari informazioni quali, il numero di tutti gli oggetti, la salute corrente del personaggio, il valore del suo scudo, e le monete in suo possesso. La funzione viene chiamata ad ogni cambio di stanza, ad ogni acquisto fatto e anche all'inizio di una **NUOVA PARTITA** (in modo da azzerare i file precedenti), in caso di morte il valore della salute viene reimpostato al valore originale prima di continuare la partita.
La funzione **loadGame** invece si occupa di assegnare i valori alle rispettive posizioni nel programma. Questa funzione viene chiamata solo nel caso in cui si selezioni continua partita nel menù principale.