A-Jump, concept di gioco e caratteristiche.

Introduzione

A-Jump è un platform il cui scorrimento della mappa avviene in maniera verticale, saltando da una piattaforma ad un'altra.

Per avanzare nella mappa è necessario che il giocatore stia attento alla piattaforma sulla quale salire: è possibile infatti incappare in brevi `vicoli ciechi`.

Protagonista e nemici possono combattere fra loro in due modi: - Contatto fisico diretto. - Sparando: player e nemici possono provare a sbarazzarsi l'uno dell'altro attraverso colpi di arma da fuoco!

Il protagonista non ama il contatto fisico, è infatti germofobico: ad un qualsiasi tipo di contatto diretto prende il covid e muore istantaneamente.

Protagonista e nemici sono dotati di health Points. Questi differiscono per ogni tipo di nemico nonchè per il player.

L'obiettivo del gioco è dunque quello di salire il più possibile nella mappa, per ogni nuovo livello in altezza che conquisterai guadagnerai un punto (altezza massima). I nemici aumentaranno, quantitativamente, man mano che si avanza nel gioco. Se gli health Points del player scendono a zero la partita termina.

Nel gioco sono presenti bonus e malus, l'interazione avviene tramite contatto diretto.

Entità in movimento

`Player:` il player è controllato dal giocatore attraverso le frecce direzionali. La barra spaziatrice lo farà sparare verso l'alto. Rappresentato in gioco dal carattere '`@`'

`Nemici:` coloro che ostacoleranno l'ascesa del nostro player; possono essere di 4 tipi.

- **Soldato semplice:** un comune soldato avente vita e danno equilibrati; probabilmente il nemico più semplice da affrontare. Rappresentato in gioco dal carattere '`¥`'
- **Tank:** come intuibile dal nome è dotato di una quantità di health Points fuori dal comune, difficile da sconfiggere ma meno temibile in quanto in grado di infliggere meno danno. Rappresentato in gioco dal carattere '`■`'
- **Artigliere:** un vero e proprio pericolo! Capace di infliggere un danno sproporzionato, l'artigliere è uno dei nemici più minacciosi. E' tuttavia sufficiente un bel respiro e del sangue freddo per sbarazzarsene, colpendolo una volta infatti morirà. Rappresentato in gioco dal carattere '`▼`'
- **Boss:** senza dubbio il più temibile. Una quantità di health Points uguale a quella del *tank* e dei danni da fuoco paragonabili a quelli dell'*artigliere* fanno di lui il nemico più difficile da affrontare. Rappresentato in gioco dal carattere '`¤`'

Entità immobili: i bonus.

Di tanto in tanto appariranno nella mappa dei particolari simboli; passandoci sopra si attiveranno rispettivi bonus o malus. Si tratta in totale di 4 simboli:

- **Bonus health points:** Questo bonus incrementa la salute del player. Rappresentato in gioco dal carattere '`♥`'
- **Malus health points:** Questo malus decrementa sensibilmente la salute del player. Rappresentato in gioco dal carattere '`∅`'
- **Bonus Bomb:** Questo bonus genera una grande esplosione che va ad eliminare tutti i nemici presenti nella mappa. Rappresentato in gioco dal carattere '`ð`'
- **Bonus Proiettili speaciali:** Questo bonus assegna al player un numero predefinito di proiettili letali. Un singolo proiettile è in grado di eliminare anche i nemici con più health points. Rappresentato in gioco dal carattere '`⌂`'

A-Jump, divisione dei compiti.

Come richiesto, la scrittura del codice è stata suddisivisa dai 4 partecipanti al progetto come segue.

- `Lorenzato Alex:` Generazione della mappa e creazione di funzioni che ne permettono la gestione. Generazione e movimento del player nella mappa.
- `Frau Alessandro:` Creazione del nemico, creazione dei proiettili, gestione della generazione dei nemici, dei proiettili, e di tutte le interazioni tra nemici, proiettili e player. Gestione dei vari thread e del flusso del gioco. Project manager.
- `Apollonio Francesco:` Creazione delle diverse tipologie di nemici e loro peculiarità. Gestione degli effetti causati dai proiettili su player e nemici Creazione dell'entità bonus, quindi algoritmo di spawn e gestione delle interazioni con player e nemici.
    - `Benatti Alice:` Gestione dell'avvio del gioco, quindi menù iniziale. Gestione della conclusione della partita, quindi salvataggio dei dati e stampa della classifica.

Scelte implementative

> Alex Lorenzato

- `Generazione mappa:`
    - la prima idea era di usare una matrice, sostituita immediatamente da una lista per risolvere il problema dei limiti della mappa; in particolare una bilista dal momento che si lavora sempre sui piani adiacenti al giocatore quindi ad esempio è molto meno dispendioso scendere di qualche piano, perché non ci sarà bisogno di scorrere la mappa dal ground floor.
    - Per tenere traccia dei piani e individuarli univocamente è stato associato un numero riga.
    - la dimensione delle piattaforme è stata decisa in modo tale da avere sempre almeno un punto di incontro tra 2 piani adiacenti
- `Movimento giocatore:` qui la scelta è stata semplice, la funzione _getch() fa da listener per i tasti, e alla pressione restituisce un intero, quindi lo switch è

risultata da subito l'opzione più comoda.

- `Stampa mappa:` all'inizio la funzione printMap stampava ad ogni "frame" tutti i piani per intero, ma a livello grafico c'erano dei problemi di visualizzazione in quanto scattava; così, grazie a delle funzioni implementate da Alessandro (move_cursor, find_char) è stato possibile leggere i caratteri direttamente da terminale e sostituire solo quelli diversi dal frame precedente (ad esempio, muovendosi in orizzontale, i piani non cambiano è non è necessario stamparli nuovamente), questo ha migliorato notevolmente la godibilità del gioco.
- `Note:` l'idea di mettere dei checkpoint sottoforma di piattaforme che coprono interamente il piano è stata sfruttata per ridurre l'eventualità di "punti ciechi".

> Francesco Apollonio

Nemici

- `Tipologia di nemico:` Ho optato per un algoritmo basato sulla probabilità. Ho impostato una probabilità più bassa per il nemico più forte e reso praticamente uguale la probabilità che spawni ogni altro tipo di nemico. Questa probabilità è stabilita attraverso la generazioni di numeri casuali. Per gestirne le diverse caratteristiche ho assegnato all'entità nemico un parametro *who_shot.* Dunque in base a questo parametro riesco a determinare la vita e il danno di ogni nemico.
- `Danneggiare un nemico:` Per fare ciò tengo in considerazione che per ogni colonna (x) ho al più un nemico. Dunque una volta sparato il proiettile se questo entra in contatto con un carattere rappresentante il nemico effettuo un check nella lista dei nemici: cerco un nemico che abbia lo stesso valore x del proiettile, ovvero che si trovi nella sua stessa colonna. Una volta trovato effettuo le dovute operazioni. (Riduzione health points / eliminazione)

Bonus

- `Generazione dei bonus:`
  - Per determinare quale bonus far spawnare utilizzo una funzione basata sulla probabilità. Questa divide esattamente in egual modo le possibilità di spawn di un bonus piùttosto che di un altro.
  - Per ogni spawn di bonus ho creato un algoritmo che partendo dal centro della mappa controlla in maniera alternata, a destra e a sinistra, se sono presenti delle basi **raggiungibili**. La prima posizione ideale, in base alle nostre condizioni, viene designata come punto di spawn del bonus.
  - Tutti i bonus sono salvati in una lista bidirezionale; in questo modo il giocatore, se non dovesse riuscire a prendere il bonus in un determinato momento, potrà successivamente tornare indietro e appropriarsene. Per evitare qualunque genere di ambiguità ogni bonus è dotato di un ID identificativo.
- `Bonus Proiettili Speciali:`
  - Gestiti attraverso la semplice creazione di un parametro. Ogni volta che il bonus viene preso tale parametro è aggiornato. Ogni volta che il player spara il parametro viene aggiornato. Se il parametro ha valore zero il proiettile sparato dal player farà danno standard.
- `Bonus Bomba:`
  - Per eseguirne l'effetto utilizzo la lista dei nemici: scorrendola elimino ogni nemico in essa presente.
- `Bonus/Malus Salute:`
  - Aggiungo/sottraggo un valore al parametro 'health_points' del player.

> Alice Benatti

- `Gestione dell'avvio del gioco:` attraverso la pressione di determinati tasti e con la funzione _getch() si naviga nelle varie schermate: quella iniziale, con il nome del gioco, quella della classifica e l'avvio del gioco.
- `Salvataggio dei dati:` grazie all'uso di una lista viene salvato e aggiornato il file della classifica, in base ai punteggi ottenuti durante la partita.
- `Classifica dei punteggi:` per realizzare la classifica ho sfruttato un file .txt in modo da non perdere i dati al momento dell'uscita dal gioco, che viene letto e salvato in una lista

> Alessandro Frau

- `Creazione del nemico e dei proiettili:` Per creare la classe nemico e proiettili ho seguito una procedura abbastanza standard, cercando di lasciarli il più estendibili possibile in modo da poterli poi espandere con le diverse tipologie assegnate poi da Francesco.
- `Gestione del movimento dei nemici e dei proiettili:` In questa fase mi sono trovato davanti a dover scegliere tra una soluzione magari più leggibile ma meno efficente ed una soluzione meno leggibile ma effettivamente decisamente più efficente, ho scelto quella più efficente per rendere il gioco più fluido. Partendo dalla soluzione scelta da me, ho in entrambi i casi deciso di creare una classe che gestisce la lista delle entita, mantenendo la lista ordinata con una priorita, nel caso dei proiettili i proiettili più in alto sono prima di quelli più in basso, mentre nel caso dei nemici i nemici più a sinistra sono prima di quelli a destra. Questa scelta permette poi, nelle fasi di movimento e interazione, di ottimizzare notevolmente il costo computazionale, passando da costi quadratici a costi lineari, o addirittura costanti in alcuni casi, ma a discapito della leggibilita del codice, infatti necessariamente gli algoritmi risultanti sono notevolmente più complessi. Una scelta sicuramente più semplice sarebbe stata quella di mantenere le liste disordinate, e di scorrerle interamente ogni volta che si necessita di effettuare una determinata azione, come per esempio eliminare un elemento, prenderne un valore o semplicemente ricercarlo.
- `Gestione della generazione dei nemici e dei proiettili:` Riguardo la generazione dei proiettili in realtà non c'è tanto da commentare, sicuramente la scelta più importante è stata quella di distinguere i proiettili del player da quelli dei nemici solo utilizzando una variabile che segna la direzione, alternativamente avrei potuto gestire i diversi proiettili in liste diverse. Riguardo la generazione dei nemici ho seguito delle regole particolari, infatti abbiamo comunemente deciso che i nemici dovessero nascere solo in colonne in cui non era presente alcun nemico, questo per evitare un effetto magari sgradevole di nemici costantemente sovrastati dai proiettili dei nemici sopra di essi, poi abbiamo sempre deciso di comune accordo di far nascere un nemico per riga **al massimo**, in modo da non esagerare con il numero di nemici presenti nella mappa allo stesso momento.
- `Gestione del movimento dei nemici e dei proiettili:` Riguardo il movimento dei nemici, è stato uno delle parti algoritmicamente più complicate in quanto per mantenere il costo computazionale basso ho dovuto fare dei passaggi particolari, infatti la necessità di non far salire due nemici nella stessa colonna e quella di far inseguire il player dai nemici hanno reso l'algoritmo decisamente più vincolante. Alcune scelte importanti sono state quella di far spostare il nemico anche in diagonale, ma mai solo lateralmente o solo verso l'alto, e quella di dare la priorita ai nemici più sotto nel caso che due nemici si vogliano spostare nella stessa colonna (i nemici si spostano sempre verso il player, quindi questa casistica avviene praticamente solo quando il player ha un nemico alla sua sinistra e uno alla sua destra). Riguardo il movimento dei proiettili, le uniche cose vincolanti legate al movimento sono state quella di fare attenzione a proiettili che salgono più in alto/più in basso di alti proiettili nella lista, e che quindi vanno spostati per mantenere i vincoli iniziali di ordinamento, e quella di sostituire in modo corretto i caratteri sopra/sotto il proiettile. Tornando indietro avrei sicuramente trovato una maniera migliore per sostituire i

caratteri, è stato troppo ottimistico da parte mia pensare di poter gestire facilmente solo con una variabile per proiettile la cosa, infatti è stata una delle parti più problematiche durante la fase di implementazione.

- `Gestione delle interazioni tra nemici, proiettili e player:` Riguardo le interazioni tra nemici e player, è bastato mettere dei vincoli sul movimento di entrambi per il controllo delle interazioni, riguardo invece le interazioni tra proiettili e player / nemici / altri proiettili, la parte di controllo della vita è stata gestita da Francesco, mentre io ho gestito tutto il resto, la parte più complicata è stata sicuramente gestire le interazioni esclusive tra proiettili che potevano avvenire in svariati modi.
- `Gestione dei vari thread e del flusso del gioco:` Si è reso necessario implementare il gioco su due thread portanti, questo perché risultava impossibile aspettare contemporaneamente la pressione di un carattere e far avvenire un animazione come quella del movimento di un proiettile. Alla fine, grazie anche alla buona implementazione di tutte le classi, il collegamento nella classe Game non è risultato complicato come mi aspettavo, l'implementazione è risultata abbastanza standard, infatti, è bastato mantenere un ordine corretto degli avvenimenti da compiere / controllare. Grazie anche alla separazione dei due thread è stato possibile incrementare la complessita del gioco in modo graduale.