Progetto Programmazione 2016-2017, Gruppo 3.

Il gioco

Il gioco consiste nel guadagnare esperienza affrontando (e sconfiggendo) i mostri che si incontreranno nel cammino.
La mappa di gioco è sviluppata su più piani con all'interno di ognuno di essi più stanze da esplorare gradualmente (verrà visualizzata solo la parte di mappa già esplorata lasciando nascosto il resto).
Il giocatore potrà spostarsi in una sola stanza per ogni turno di gioco.
Se incontra un mostro il combattimento è obbligato (non si può fuggire né quando si è nella stessa stanza né una volta iniziata la battaglia).
Si vince quando il giocatore raggiunge il livello 20 di esperienza.
Si perde morendo durante un combattimento perché sconfitti dal mostro.

La classe **mappa**

Per ogni piano viene generata una mappa con la creazione di una classe "**mappa**". Le classi di ogni piano sono collegate in una lista di puntatori ad ognuna di esse (`struct lista_piani`).

La classe **`mappa`** contiene al suo interno la matrice di puntatori alle stanze, il piano della mappa, e vari parametri che indicano la dimensione della matrice, il numero delle stanze, nonché i metodi che saranno sfruttati dal costruttore, dal main e dalle altre classi. Le stanze sono implementate come strutture con all'interno le coordinate della stanza (`int x, y`), il contenuto (`int contenuto`), i puntatori alle stanze adiacenti (`ptrstanza w, d, s, a`), a quella precedente (`ptrstanza precedente`) e le scale (`ptrstanza scale`), che sono diverse da NULL solo nella prima stanza di un piano superiore allo 0, e nell'ultima stanza di ogni piano.

Il numero di stanze per ciascun piano è il numero del piano a cui viene sommato 6.

La matrice è quadrata e, per avere delle dimensioni contenute, esiste un controllo (`stessa_direzione`) che fa sì che non vengano costruite stanze più di tre volte di seguito sulla stessa riga/colonna. Pertanto la funzione `dimensione` trova una relazione tra il piano e la dimensione della matrice in modo che sia adeguata per contenere tutte le stanze senza eccessivi sprechi di spazio.

Questa matrice, una volta calcolatene le dimensioni, verrà inizializzata tramite `inizializza_matrice` che crea una matrice delle dimensioni volute con puntatori alle stanze (`ptrstanza`) inizializzati a `NULL`.

Il costruttore prende in input il piano e un puntatore alla stanza precedente, che può essere `NULL` nel caso in cui siamo nel livello 0.

La creazione della mappa avviene creando la prima stanza nelle coordinate (0,0), dopodiché a questa viene collegata una nuova stanza in una delle quattro porte possibili: `w` (puntatore a nord), `d` (puntatore a est), `s` (puntatore a sud) e `a` (puntatore a ovest). Prima di inizializzare tali porte, vengono fatti tre controlli: che la porta che si vuole inizializzare non si apra oltre il bordo della mappa (`is_border`), che non sia già stata costruita una stanza in quelle coordinate (`gia_costruito`) e che non siano state costruite troppe stanze nella stessa direzione (`stessa_direzione`).

Dopo aver controllato in quali direzioni sia possibile costruire, una funzione *random* sceglie in quale stanza effettivamente costruire. Nel caso in cui restino ancora stanze da costruire nella mappa del piano in cui si sta lavorando e, rispetto ai controlli sopra citati, risulti impossibile costruire nelle quattro direzioni, si torna alla penultima stanza costruita e lì si rifanno i controlli sulle varie porte. Se ancora questa non da alcun risultato positivo, si torna indietro e così via, finché non si trovi una porta che superi i controlli per la costruzione.

Ogni volta che viene generata una stanza, questa viene inserita in una lista (`elenco_stanze`) che ne memorizza il puntatore e le coordinate. Questa torna utile nella parte finale del costruttore, quando viene invocata la funzione `riempi_stanze()` che sceglie in maniera randomica, tra le stanze costruite, `mostrixpiano` stanze in cui inserire i mostri.

Ogni cinque piani, inoltre, a partire dal piano 0, il contenuto dell'ultima stanza verrà *settato* al numero corrispondente ad uno dei quattro *boss* creati, in base al piano in cui ci si trova.

Ogni volta che viene scelta una stanza in cui inserire il mostro, le sue coordinate, il suo tipo, nonché il puntatore alla stanza in cui si trova, vengono inserite in una lista, che il **game manager** utilizzerà per creare i mostri come classi.

La stampa della mappa

All'interno della classe **mappa**, c'è un metodo che permette di stampare la mappa aggiornata grazie alle coordinate del personaggio che vengono date dal game manager ogni volta che viene invocata la stampa, nonché all'accesso al contenuto delle stanze che rivela la presenza di un mostro, stampando questi nelle stanze opportune. Il metodo `stampa_mappa(int x_p, int y_p)` va a creare un array lungo quanto un lato della matrice che andrà a riempire con degli 1 nel caso siano presenti dei collegamenti tra le stanze, sfruttando le funzioni ausiliarie `check_links`, `check_linkd` e `mid_line`. Scorrendo poi prima sulla y, poi sulla x, la `stampa_mappa` stampa una stanza laddove presente,

inserendo una “**P**” nel caso sia presente il personaggio (le cui coordinate sono date come argomento del metodo), una “**B**” nel caso sia presente il Boss, una “**M**” nel caso sia presente un mostro diverso dal boss, “**#**” nel caso siano presenti le scale per salire al piano successivo, o dei tab (`\t`) nel caso non sia presente nessuna di queste cose.

Oltre alla classe **mappa** abbiamo le classi **`bot`**, **`player`** e **`mostri`**.

Le classe **`bot`**, e le sue sottoclassi **`player`** e **`mostri`**.

La classe **`bot`** è la classe genitore di player e mostri. Essa contiene le variabili fondamentali che verranno usate nelle due classi, un costruttore nullo, che serve soltanto per evitare di ripetere il codice per le due classi, nonché piccole funzioni di ritorno che che servono a proteggere le variabili `protected` di **`bot`**, un metodo per la stampa delle statistiche del combattimento e funzioni di controllo che aumentano o diminuiscono le statistiche del giocatore e dei mostri durante il combattimento in base alle *abilità* usate durante di esso, controllando quali, tra quelle fornite implementate, sono effettivamente utilizzabili in quel punto.

La classe **`player`** è la classe con cui verrà creato il personaggio che verrà comandato dal giocatore. Essa eredita da **`bot`** le variabili di attacco, gestione dei turni ed esperienza, a cui vengono aggiunte le variabili relative alle abilità e il loro tempo di ricarica (con i turni come unità di misura). Il costruttore di **`player`** inizializza le statistiche del personaggio in modo adeguato, evitando squilibri sia a vantaggio del personaggio che del mostro. Inoltre il costruttore inizializza le variabili relative al personaggio in modo tale che non tutte le abilità siano immediatamente disponibili, ma vengano sbloccate con l’aumento del parametro relativo all’esperienza. Tra i vari metodi anche quelli per accedere a variabili `protected`, incluso il metodo per aumentare il *livello* del giocatore, al cui aumento sono associati la possibilità di sbloccare nuove abilità che determineranno diversi stili di combattimento, rendendo il gioco più strategico che aleatorio. Infine i due metodi `movimento` e `turno_p`, rispettivamente, modificano le coordinate di posizione del giocatore e gestiscono i turni del `player`, che in combattimento potrà scegliere le abilità da usare.

`mostri`, invece, è la classe che permette di creare i mostri laddove la mappa ne rivela la presenza e di permetterne l’autogestione durante la fase di combattimento tra il *player* ed essi.

Oltre alle variabili fondamentali della classe **`bot`**, **`mostri`** contiene le variabili legate al tempo di ricarica delle mosse dei mostri e una variabile che indica il tipo di mostro a cui ci si sta riferendo: si possono incontrare infatti sedici tipi di mostri, divisi in quattro categorie (*animali, scheletri, essere celesti, demoni*) in

ognuna delle quali è presente un mostro con le potenzialità impostate ad un valore più alto, che chiameremo *boss.*

La funzione `turno_m`, che gestisce il turno dei mostri, è molto simile a quella del giocatore, ma contiene delle piccole modifiche che permettono di autogestire il mostro e di conferire un certo vantaggio al boss.

Infine il metodo `azione` contiene tutte le possibili mosse che un mostro può compiere e, in base alla variabile che definisce il tipo di mostro, effettua la mossa adeguata ad esso.

Il *game manager*

Il *game manager* (`main`) è organizzata secondo due `while` principali annidati: il primo che corrisponde alla serie di piani (che proseguirà fino a quando la partita non sarà finita - quindi alla vittoria o morte del giocatore) e il secondo che corrisponde alla serie di stanze all'interno di ciascun piano.

Risulta necessario l'utilizzo di due variabili di controllo che permettono di definire i vari casi in cui possiamo trovarci: la prima è il booleano `dietro` che, se è uguale a `true`, indica che il giocatore ha appena deciso di prendere le scale per tornare al piano precedente; la seconda è l'intero `piano_max` che corrisponde al piano massimo esplorato (una volta che si accede al piano successivo a `piano_max` essa non viene subito aggiornata, potendo così sfruttare il fatto che in quel momento `p` è maggiore di `piano_max`, per poi essere posta uguale al valore di p appena prima di accedere alla funzione `piano` successiva).

Ogni piano viene definito assegnandogli la rispettiva mappa: creandola (e aggiungendola alla lista dei piani) se il giocatore accede a quel piano per la prima volta, cercandola nella lista dei piani se invece è già stato in quel piano almeno una volta.

La lista dei piani di gioco (`lista_piani`) sfrutta puntatori a classi **mappa** (`ptr_mappa`). Si accede in testa e si inserisce in coda.

Dopodiché si passa alla generazione dei mostri descritti nell'elenco dei mostri relativi al piano corrente passato in input dalla classe mappa appena creata. I mostri saranno aggiunti (`inserisciMostri`) alla lista (`lista_mostri`) che conterrà così anche i vecchi mostri non ancora uccisi.

La lista dei mostri ancora in vita, relativa a tutti i piani esplorati, sfrutta puntatori a classi mostri (`ptr_mostro`). Si accede e si inserisce in testa per poter raggiungere più velocemente i mostri inseriti più di recente.

Nel caso in cui il giocatore stia tornando al piano precedente (`dietro = true`) vengono modificate le sue coordinate (`mod_coo`) per porle uguali a quelle

dell’ultima stanza del piano a cui sta accedendo (corrispondenti alle scale da cui sta scendendo).

Viene così stampata la mappa per visualizzare il piano corrente in cui si trova il giocatore (che si troverà nella prima o nell’ultima stanza a seconda che stia scendendo o salendo nei piani).

Si passa poi all’interno della funzione piano, quindi alla gestione delle stanze in cui vengono considerati il movimento e il combattimento del giocatore.

Questa funzione infatti riceve input dalla mappa tramite un metodo le direzioni, in cui è permesso al giocatore accedere, delle stanze adiacenti a quella in cui esso si trova.

Inizialmente controlla se il giocatore, nel caso in cui si trovi nella prima (`tornaIndietro`) o nell’ultima stanza (`chiediScale`) di un piano, voglia salire le scale.

Se il giocatore decidesse di salire al piano successivo verranno modificate le sue coordinate rendendole uguali a zero (`azz_coo`).

In caso contrario si avvia la naturale gestione del turno di gioco che comprende il movimento nelle stanze consentite dalla posizione corrente del giocatore e dalla struttura della mappa del piano.

Effettuato lo spostamento viene verificata la presenza di un mostro (controllando la lista dei mostri) all'interno della nuova stanza e in tal caso si procede con il combattimento che si concluderà necessariamente con la morte di uno dei due.

Se a morire è il mostro esso viene rimosso dalla lista dei mostri (`rimuoviMostro`).

Se a morire è il giocatore allora si pone il booleano `fine` uguale a `true`.

Una volta terminato il combattimento si stampa nuovamente la mappa senza il mostro appena ucciso.

A questo punto si controlla se il giocatore ha vinto (cioè se il booleano `fine` è uguale a `true`) uscendo in questo modo da tutti i cicli e ponendo fine alla partita.