Università degli Studi di Bologna

Programmazione ad Oggetti

“DESCENT”

Realizzata da:

Corrado Stortini

Francesco Carlucci

Jonathan Lupini

Lorenzo Todisco

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il software Descent mira alla realizzazione di un videogioco rogue-like, senza trama, con dei livelli di default ma con la possibilità di aggiungere livelli custom.

Il personaggio, controllato dal giocatore, dovrà andare in giro per la mappa uccidendo i nemici e raccogliendo la chiave per andare al livello successivo.

All'inizio il personaggio parte con una quantità di vita e di energia fisse, se perde tutta la vita, il personaggio muore ed è game over, a quel punto il giocatore può tornare al menu principale o chiudere il gioco.

L'energia viene consumata per usare le abilità che il personaggio sblocca a certi livelli e si ricarica nel tempo. Il personaggio parte dal livello uno e può salire di livello uccidendo i nemici, fino ad un livello massimo.

Il giocatore può usufruire di armi, accessori e pozioni che trovano in giro per la mappa che gli permettono di superare il livello più facilmente.

Requisiti funzionali:

- All'avvio del videogioco, verrà mostrato il menu principale per poter iniziare una nuova partita sia con livelli di default che con livelli custom, aprire le impostazioni e uscire dal gioco.
- Le partite si svolgono all'interno delle mappe in cui si viene attaccati dai nemici e si deve cercare di sopravvivere e trovare la chiave per passare al livello successivo.
- Il giocatore avrà una barra della vita che una volta finita lo porterà allo schermo di game over.
- Il giocatore avrà anche una barra del mana, con cui può usare abilità per sopravvivere più facilmente.
- I nemici inseguono il personaggio principale fintanto che hanno visione di esso, cercando di ucciderlo a tutti i costi.
- Il giocatore dovrà cercare di trovare la chiave per passare al livello successivo.

- All’ultimo livello, il giocatore dovrà uccidere tutti i mob per poter raccogliere la chiave.

Requisiti non funzionali:

- Performance accettabile su vari tipi di sistemi operativi.

1.2 Analisi e modello del dominio

Per raggiungere il proprio obbiettivo, il giocatore potrà eseguire le seguenti azioni: muoversi, attaccare i nemici, raccogliere e usare gli oggetti. Quando il personaggio non ha più punti vita il gioco termina.

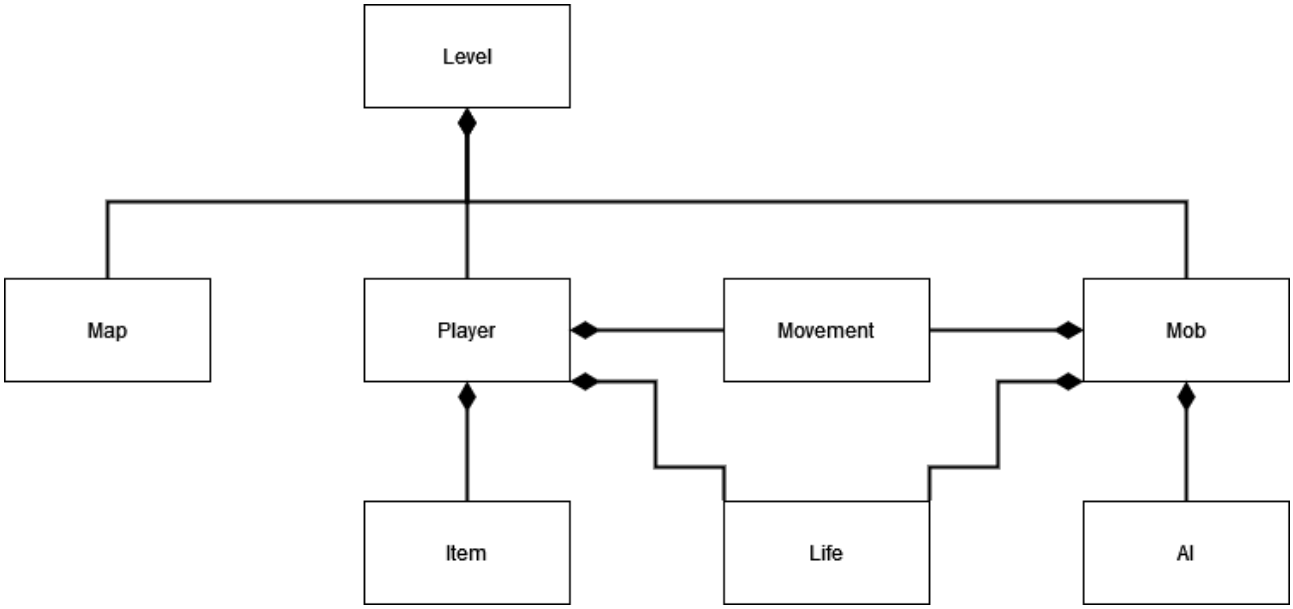


Fig.1.1 Schema UML dell’analisi del dominio, con rappresentate le principali entità e le relazioni tra di loro

Si voleva sviluppare anche un menu dei settings per audio e difficoltà, ma per motivi di tempistica, non è stato possibile.

Capitolo 2

Design

2.1 Architettura

Si è scelto di usare il framework LibGDX per i seguenti motivi:

- Grandi disponibilità di documentazione online
- Forte integrazione con altri third party program (es. TiledMap) usati
- Alta portabilità
- Molte opzioni per la gestione delle finestre (resize, zoom…)
- Potente motore grafico e audio
- Gestione input da vari dispositivi (tastiera, mouse, touch screen…)
- Open source

L'architettura del videogioco segue il pattern MV, dato che i vincoli della libreria LibGDX non consentono un adempimento totale del pattern MVC. Ad esempio, il movimento è vincolato al metodo render() presente nella classe gameScreen(). La maggior parte della view e controller è rappresentata da gameScreen() che chiama le view delle varie classi a cui è associato il model (per esempio, chiama la HeroView che sarebbe la view della classe di model Hero).

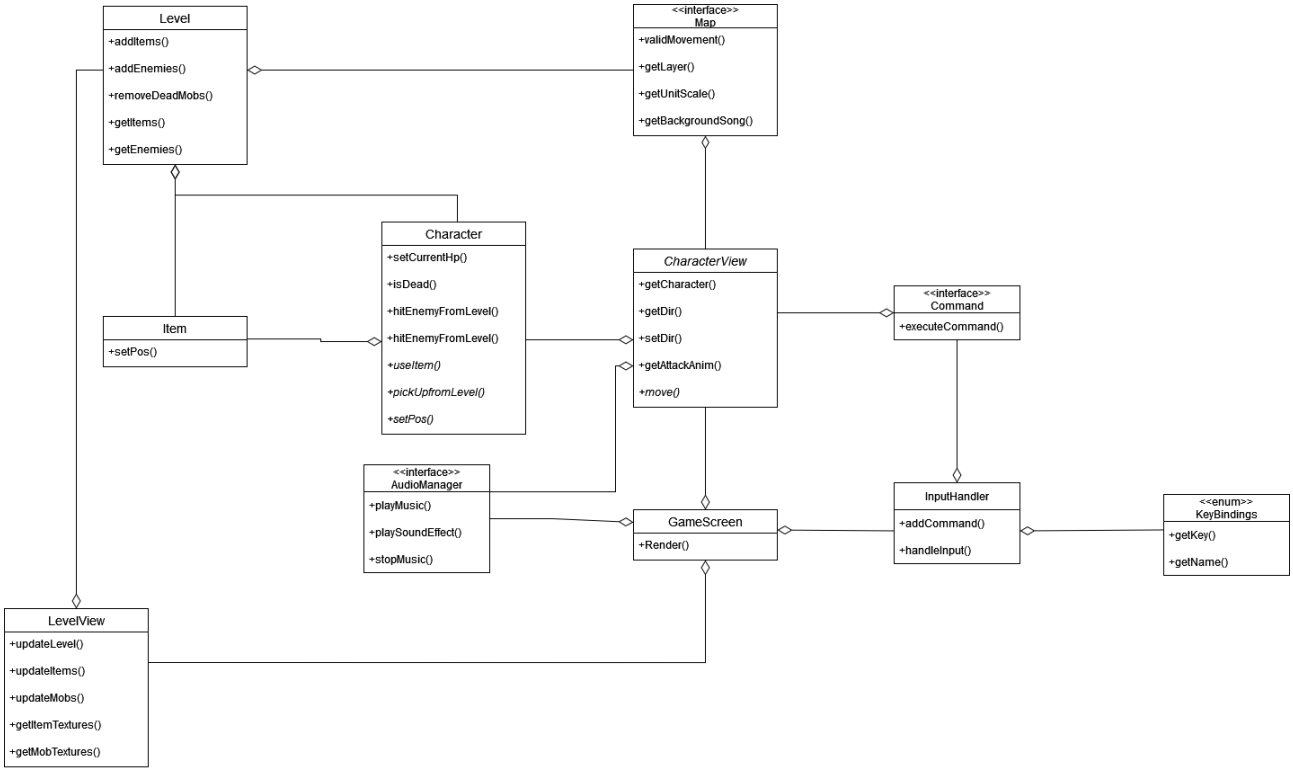


Fig. 2.0 Schema UML dell'architettura generale del videogioco

2.2 Design dettagliato

Corrado Stortini – Command/KeyBinding

Command è una funzionalità che permette ad un giocatore o ad una IA di eseguire un comando su un personaggio del gioco, qualunque esso sia, mob o personaggio principale.

Tale funzionalità è stata sviluppata in modo tale da permettere di creare nuovi comandi quando e come si vuole, ha quindi una alta scalabilità.

L'architettura utilizzata (Figura 2.1) per la realizzazione è l'omonimo pattern *Command*, studiato attraverso il libro "*Programming Game AI by Example*" di *Mat Buckland*. Questo pattern permette di implementare anche in maniera veloce ed efficare la gestione del KeyBinding, ulteriore motivo per cui è stato scelto di utilizzarlo.

Per far ciò, si è utilizzata l'interfaccia Command, l'enumeratore KeyBindings e la classe InputHandler.

Command è un'interfaccia funzionale, il che vuol dire che il metodo executeCommand potrà essere completato ogni qual volta che un comando viene creato, oppure può ovviamente essere implementata attraverso una classe o estesa attraverso un'altra interfaccia. Tale comando sarà eseguito su CharacterView, da cui si piò risalire a Character o Hero (o HeroView).

L'enumeratore KeyBindings ha una lista di coppie (*chiave, nome*), dove la *chiave* rappresenta il tasto da premere sulla tastiera per eseguire il *comando* nominato con *nome.* Sarà poi l'InputHandler ad associare ad ogni coppia il *comando* da eseguire, affinchè possa poi eseguire un certo *comando* quando la rispettiva *chiave* viene premuta.

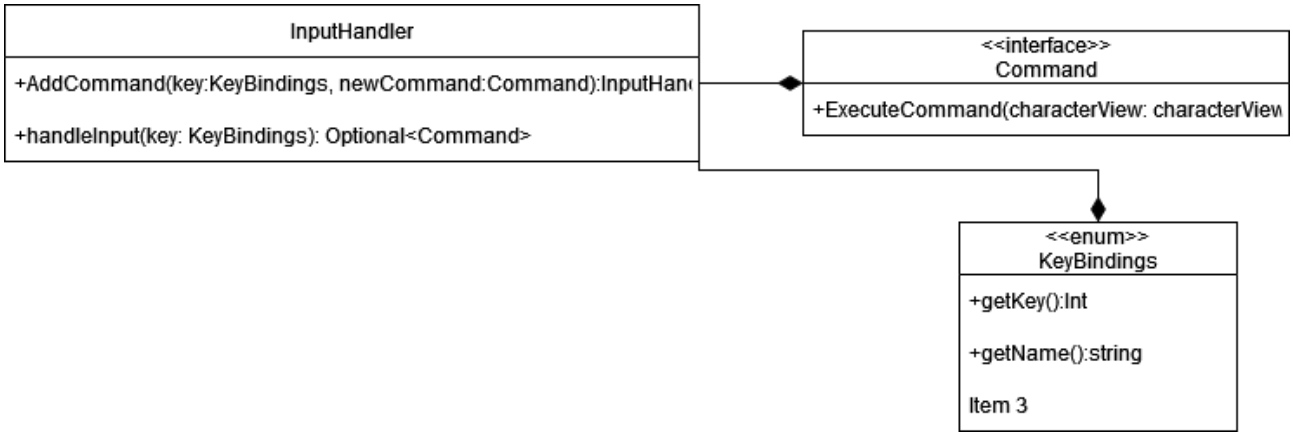


Fig. 2.1 Schema UML che rappresenta l'interfaccia Command che viene utilizzata attraverso un inputHandler

Per implementare alcuni comandi più complicati è stato necessario creare classi che estendono l'interfaccia *Command*, come Movement e Skill.

Movement è una classe che permette di eseguire un movimento in una delle quattro direzioni (Left, Right, Up, Down) in base alla direzione passata in input.

Skill è una classe astratta che permette la gestione di una skill del personaggio o del mob, ovvero una abilità speciale che consuma energia (o mana).

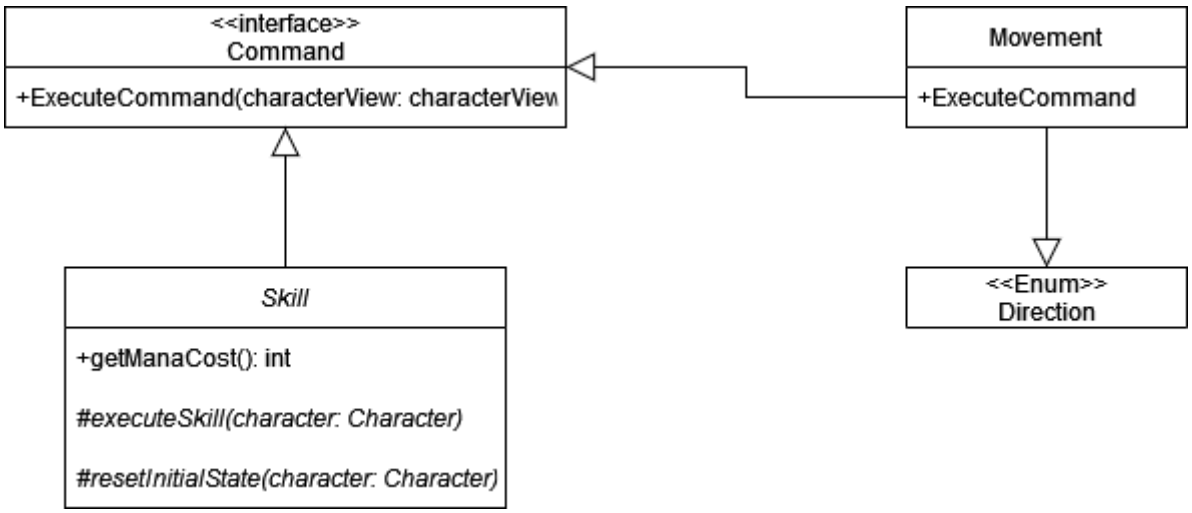


Fig. 2.2 Schema UML che mostra come sono implementate le skill e i movimenti

Corrado Stortini – Mob

I mob sono i nemici del gioco, pertanto sono esseri "viventi" tanto quanto il personaggio principale, questo vuol dire che ognuno di loro avrà delle statistiche diverse in base alla razza di provenienza.

Per implementare quanto scritto sopra, si è deciso di utilizzare una class di enum ModStats, in cui le varie razze di nemici sono descritte attraverso un nome, una vita, una velocità di movimento, del mana e dell'esperienza che daranno al personaggio principale.

Tale enumeratore verrà poi utilizzato dalla classe Mob, estensione della classe astratta Character, che si occuperà di creare i mob in base alla loro razza e ad un arma passata in input.

Sarà poi compito della classe MobView di gestire la view del Mob.

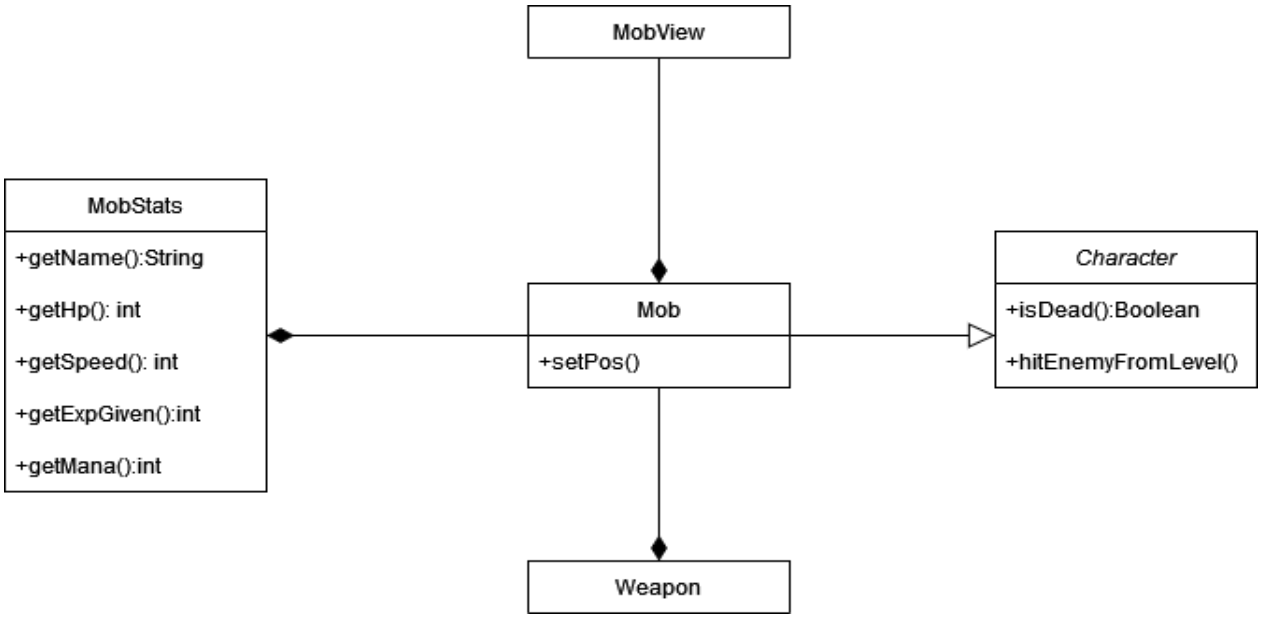


Fig. 2.3 Schema UML che rappresenta l'architettura usata per modellare i Mob

Corrado Stortini – Level Management

La funzionalità di Level Management (gestione del livello) è stata implementata in Hero, ed è essenzialmente semplice: quando un mob muore, dell'esperienza viene aggiunta all'eroe, e quando l'esperienza arriva ad un certo limite, l'eroe fa il level up, aumentando la quantità di esperienza necessaria per livellare ancora, e con la possibilità di sbloccare una nuova abilità a certi livelli.

Francesco Carlucci – Menu

I menu sono elementi fondamentali all'interno di una applicazione per permettere all'utente di navigare tra le varie possibilità di cui dispone.

Appena aperta l'applicazione ci si trova davanti il menu principale che permette di avviare il gioco vero e proprio.

La libreria LibGDX permette di creare le varie schermate di un'applicazione come implementazioni dell'interfaccia Screen, fornendo i metodi necessari per rimodellare la view in base alla finestra dell'applicazione e per renderizzarla, di conseguenza i menu e la schermata di gioco implementano questa interfaccia.

Dal menu principale si può accedere al menu delle impostazioni e viceversa, mentre una volta terminato il gioco si accede al menu di GameOver che permette di uscire dal gioco o tornare al menu principale.

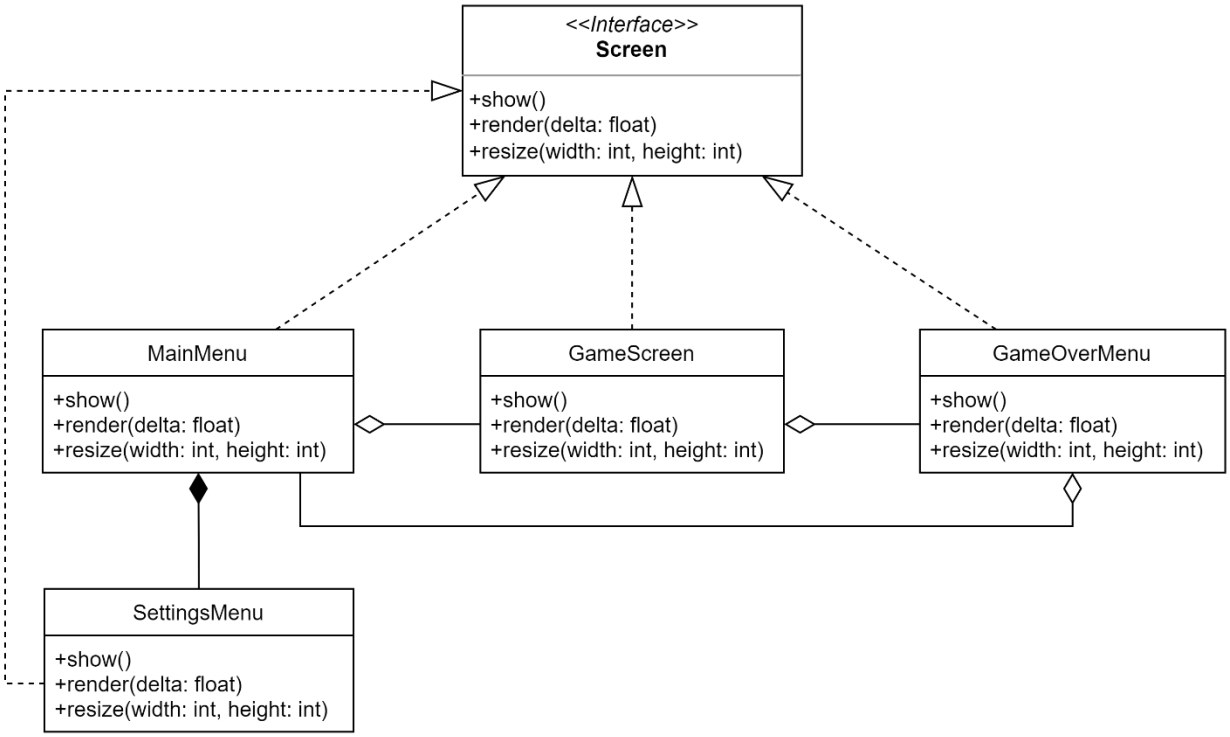


Figura 2.4 Schema UML dei menu

Francesco Carlucci – Gestione dei livelli

Dal momento che il gioco è costituito da più livelli è stato necessario implementare un sistema per gestirli in maniera corretta.

La soluzione consiste nell'implementare il gestore dei livelli come una classe, contenente una lista ordinata, che tiene il riferimento del livello corrente tramite un contatore.

Gli elementi della lista sono i veri e propri livelli che costituiscono il gioco, ognuno personalizzato con i propri oggetti, nemici e mappa.

Tramite la suddetta classe si verifica anche il completamento del gioco da parte dell'utente, ovvero quando non ci sono più livelli mancanti e il giocatore usa la chiave per aprire la porta il gioco termina.

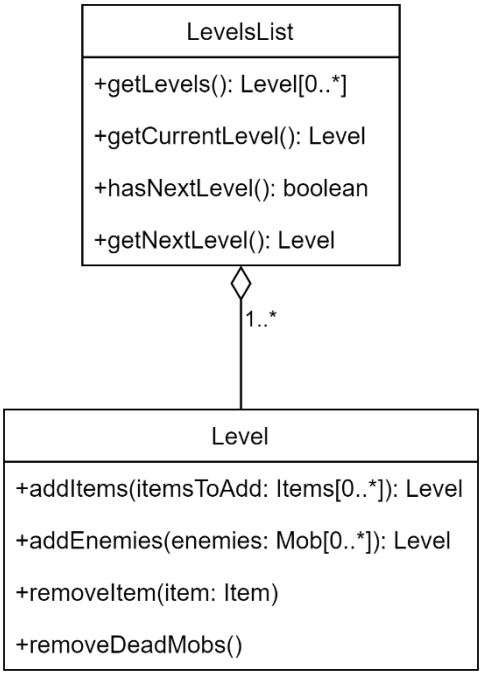


Figura 2.5 Schema UML dei livelli

Francesco Carlucci – Oggetti Indossabili

Nella mappa sono presenti vari tipi di oggetti, tra questi ci sono gli indossabili (es. anelli, amuleti, ecc.) che danno al giocatore determinati benefici.

Per crearli è stato utilizzato il pattern Builder che permette di istanziare differenti tipi di indossabili senza dover coinvolgere numerosi costruttori o classi ad hoc per ciascun tipo.

Inoltre, attraverso questo pattern, è possibile mischiare le proprietà che si vogliono conferire agli oggetti consentendo la massima personalizzazione per ciascun indossabile.

Un ulteriore motivo per cui è stato utilizzato questo pattern è la possibilità di utilizzare un'interfaccia fluent, in cui i metodi del builder ritornano "this", rendendo il codice chiaro e leggibile.

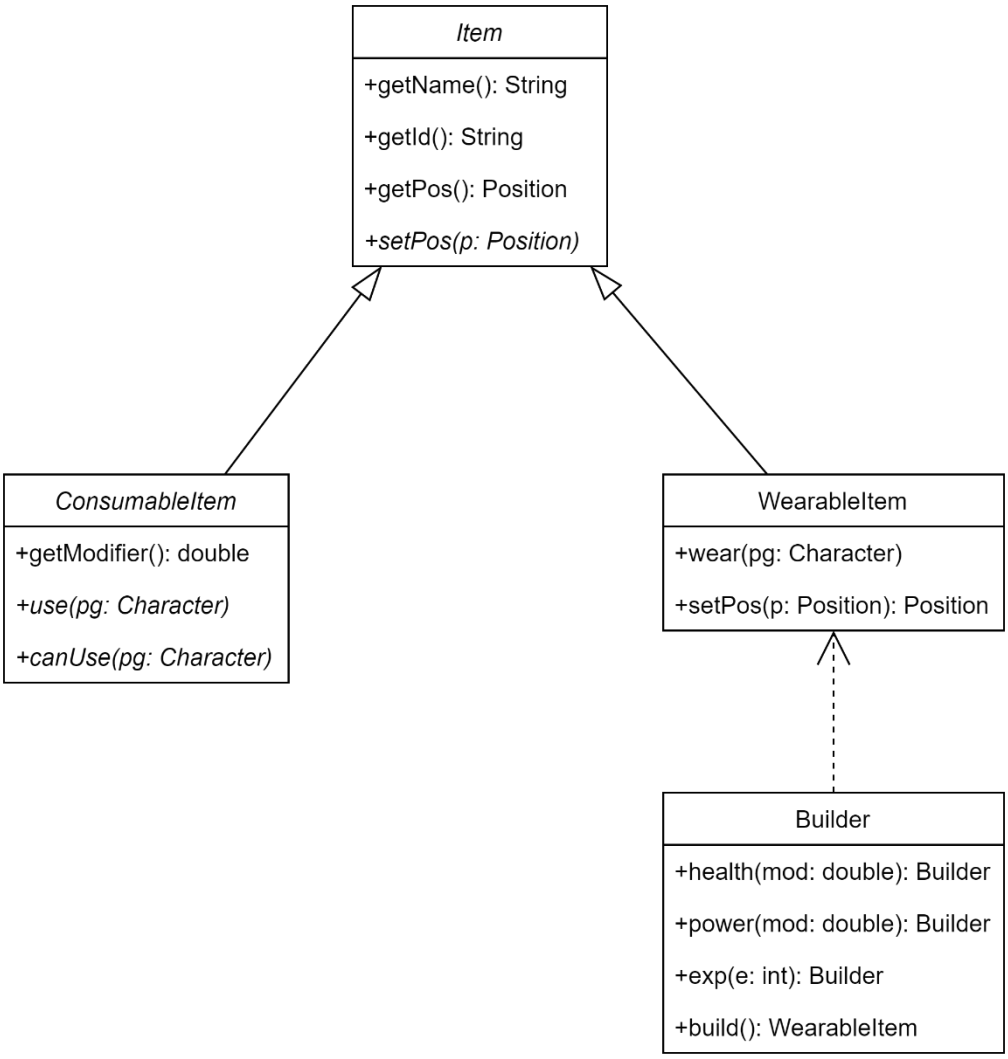


Figura 2.6 Schema UML degli oggetti indossabili (wearable)

Jonathan Lupini – Mappe

È necessaria una premessa, in Descent la mappa è un'entità di model che gestisce l'esistenza di pareti e tile speciali quali trappole, teletrasporto etc. Le entità del gioco (item, personaggi) risiedono nel Level, non nelle mappe.

Sin da subito si è provata necessaria l'esistenza di un'interfaccia: le mappe da noi usate sono fatte a mano e quindi non fruibili per una potenziale modalità infinita, e il nostro mondo è fatto di unità, non di tile (che sarebbero potute servire in livelli puzzle like). Ho quindi optato per definire un'interfaccia generica DescentMap che dichiarasse metodi comuni alle mappe create con Tiled, in modo da poter definire varie implementazioni nel caso avessimo creato più modalità.

La mappa è puramente model, contiene riferimenti alla musica di sottofondo e alla parte grafica della mappa ma sono gestiti entrambi dal GameScreen che in questo caso fa le veci del controller.

Nell'implementazione attuale ogni mappa ha vari layer, uno per l'immagine effettiva della mappa e uno per poligoni che fanno le veci di pareti e oggetti, e vari layer per aree speciali definite sempre con poligoni. Tramite metodi forniti da libgdx si confronta la lista di poligoni ottenuta dai layer con l'hitbox di un personaggio calcolata nella posizione verso cui vuole andare, se sono sovrapposte si verifica una collisione o l'attivazione di aree speciali (a seconda del layer da cui si sono presi i poligoni), questo tramite i metodi validMovement, checkTeleport e checkDamageTile, chiamati dal GameScreen e da Command quando un Personaggio si muove.

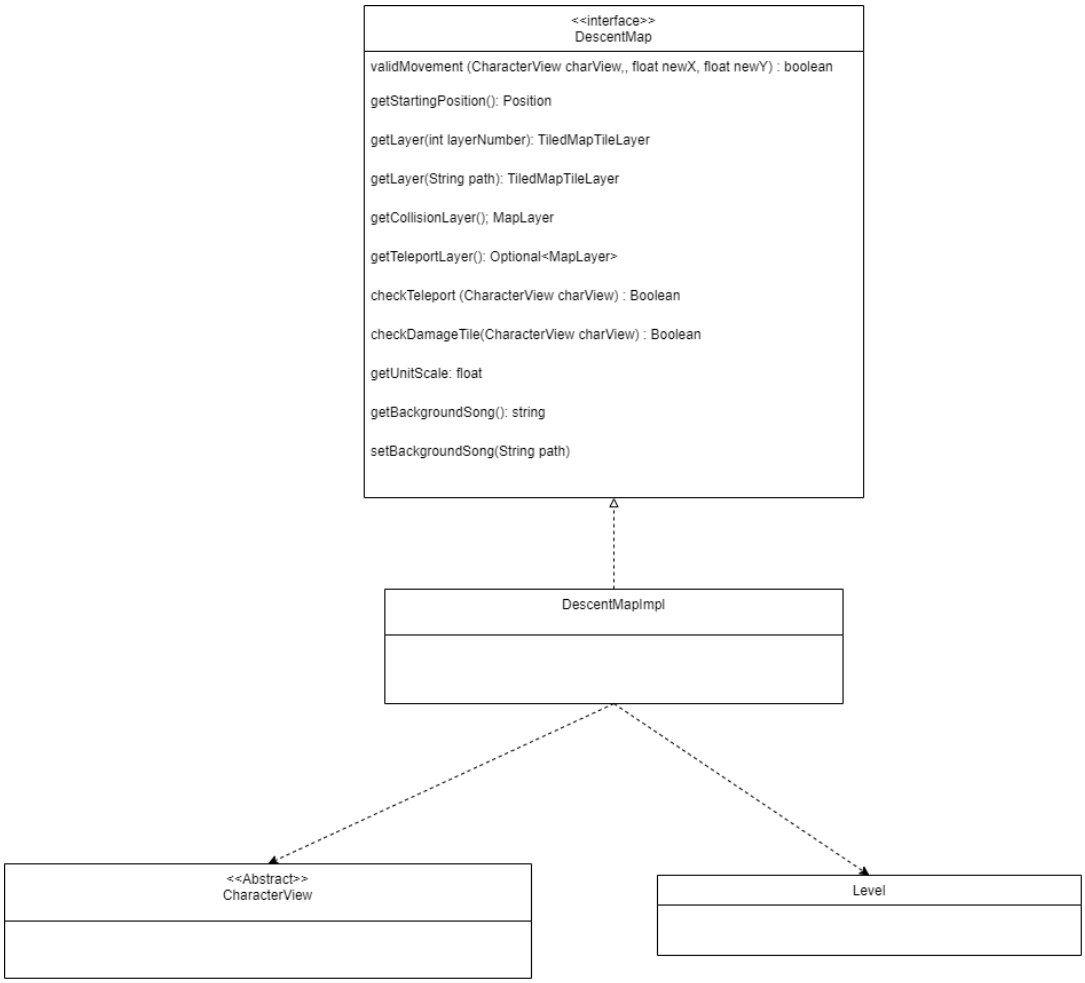


Fig. 2.7 Schema UML che rappresenta l'architettura e l'uso delle mappe

Jonathan Lupini – AI

Dal momento che le uniche entità dotate di AI nel gioco sono i mob è solo la mobView ad interfacciarsi con le classi del package AI, ho scelto di usare la view dei mob invece che il loro model (mob) perché nella nostra implementazione il movimento dei personaggi è legato alla grafica. Non avendo un controller dedicato tutti i metodi di AI partono a livello view e poi estraggono il model dai campi della view.

Pathfinding:
Attraverso il pattern strategy ho astratto il pathfinding dei mob nell'interfaccia pathfinding in modo da poter una stessa MobView ma con algoritmi diversi, questo per permettere riutilizzo nel caso di diversi livelli di difficoltà (con mob sempre piu intelligenti) o algoritmi migliori nel caso di aggiunta di nodi/tile alla mappa.

Ho optato per strategy invece che classe utility perché è il metodo moveMob ad aggiornare la posizione del mob, cosa che andrebbe oltre i compiti di una classe utility. Strategy permette anche di cambiare facilmente tra più implementeazioni visto che la sintassi dentro a mobView dipende dall'interfaccia e non dall'implementazione. Pathfinding contiene un metodo statico per ottenere una direzione random, ho optato per interfaccia con metodo statico invece che classe astratta o classe utiliy perché il metodo non ha bisogno di una classe e il numero di classi e package è già elevato.

Il metodo update() della MobView controlla se l'eroe è in raggio d'attacco, poi decide se attaccare col metodo attack() o far muovere il mob tramite l'implementazione di pathfinding. Il fatto di aver separato l'attacco e il pathfinding permette di riutilizzare mobView per mostri amichevoli evitando di attaccare.

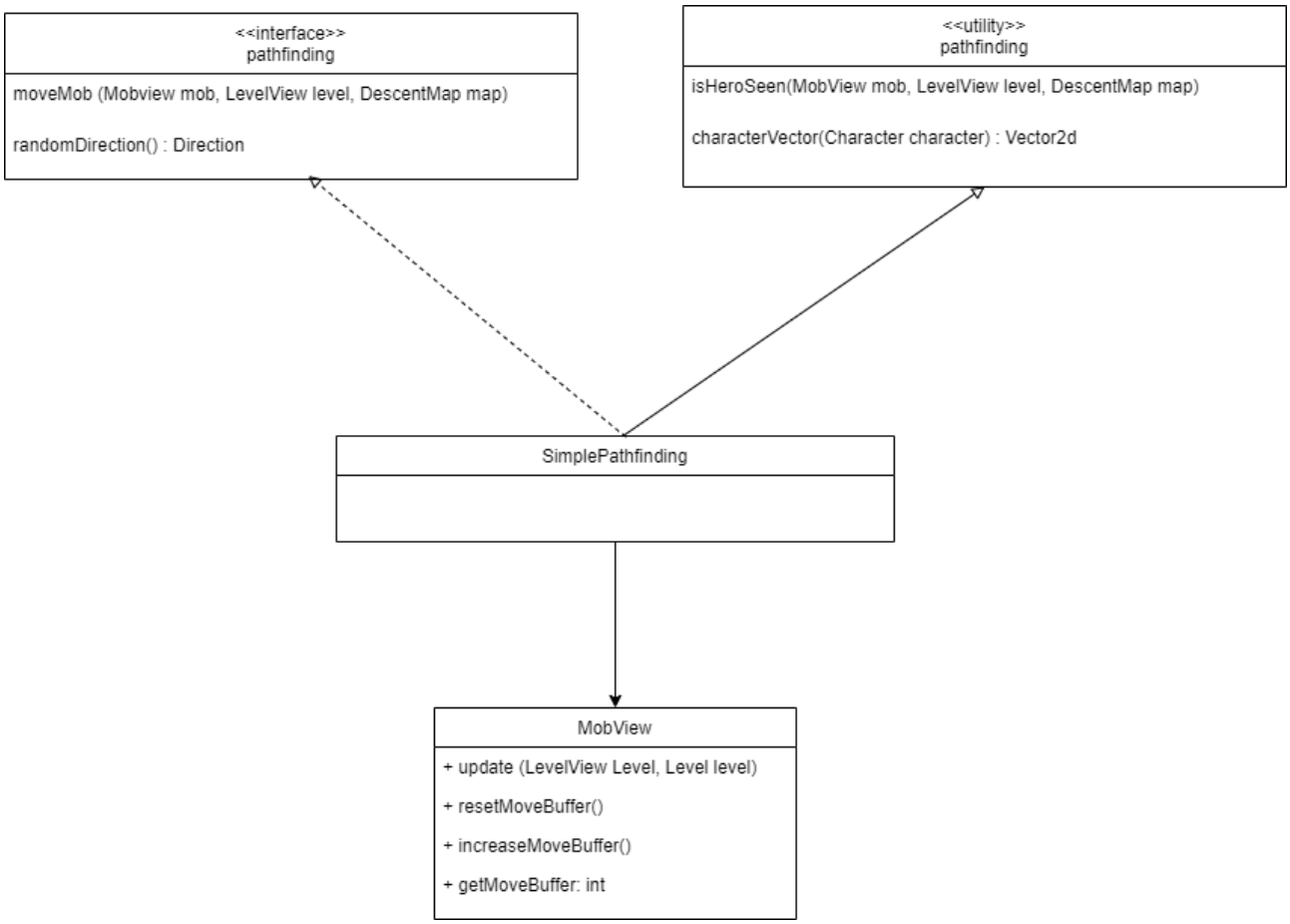


Fig. 2.8 Schema UML che rappresenta l'architettura usata per l'AI dei mob

Jonathan Lupini – Audio

Per aderire al single responsibility principle e DRY ho optato per adoperare un sistema observer-like e un'interfaccia per l'AudioManager.
Personaggi e mappe mantengono internamente, tramite costanti, riferimenti al filepath dell'audio che vogliono usare. Quando accade un determinato avvenimento, quale ad esempio il muoversi del personaggio, la classe che gestisce l'avvenimento prende quel filepath e lo manda all'audioManager, l'audiomanager riceve la notifica e a seconda del tipo di audio e delle impostazioni (E.G. volume, looping) lo fa partire, le classi possono in seguito alterare o fermare l'audio mandato. L'avere un'interfaccia per l'audioManager permette di cambiare facilmente tra vari manager a runtime, ad esempio un'esplosione potrebbe aggiunere un tintinnio ai suoni percepiti dall'eroe o un livello subacquatico potrebbe aggiungere riverbero. Permette anche di cambiare facilmente libreria audio nel caso i limit di quella di libgdx, dettati dall'integrazione con sistemi android, siano troppo restrittivi.
L'audiomanager in Descent è unico, viene creato dal GameScreen e poi dato a tutti gli oggetti che lo richiedono.

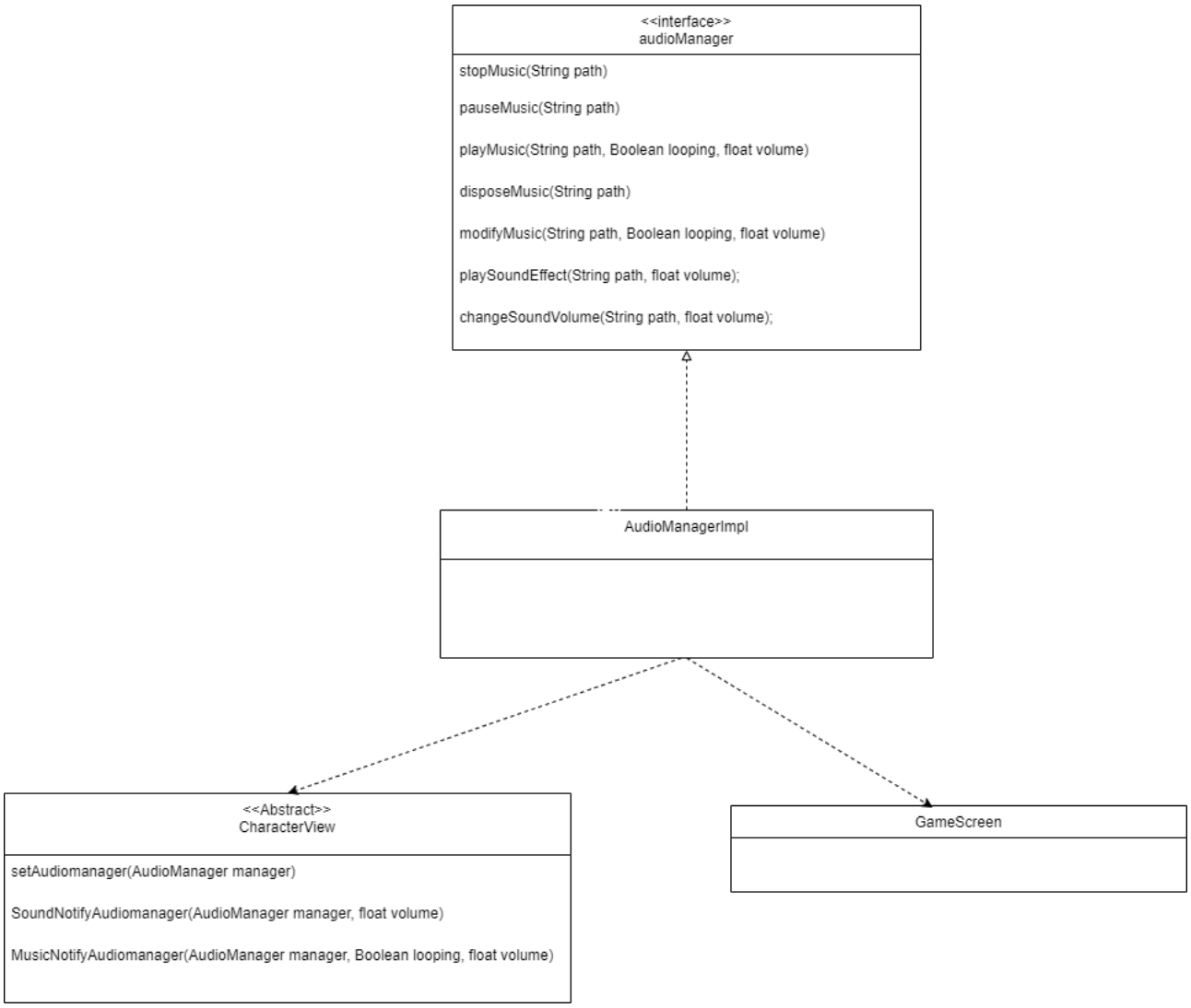


Fig. 2.9 Schema UML che rappresenta l'architettura usata per la gestione dei Suoni

Lorenzo Todisco

Il mio compito è stato creare model e view delle parti principali riguardanti gli item e i personaggi, le classi di utility per caricare i livelli a partire dalle informazioni nei file .txt, il menù di pausa e la classe Position, usata in tutte le entità.

La classe Position è piuttosto semplice, è composta di due campi per le coordinate e setter e getter; tuttavia, è fondamentale per la gestione del mondo di gioco in quanto collega i vari personaggi con gli item e il mondo di gioco.

Il Menù di pausa è un menù grafico che mette in pausa il gioco e permette di tornare al menu principale, chiudere il gioco o riprendere di giocare.

Per l'implementazione degli item ho fatto ricorso al pattern Template Method in modo da creare uno "scheletro" per i metodi in comune ed evitare di riscrivere codice. Tutti gli item presenti in gioco estendono da Item ma sono raggruppabili in due macrocategorie, consumabili e non.

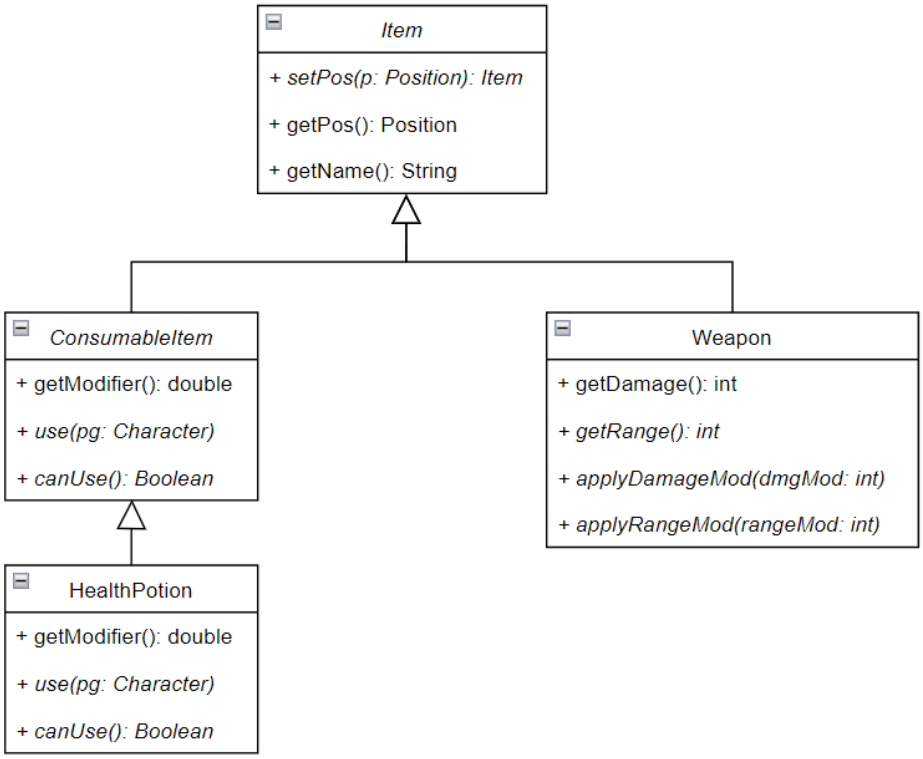


Figura 2.10 Schema UML degli item (HealthPotion e Weapon)

La classe LevelView è responsabile del caricamento degli assets riguardanti items, mobs e barre della vita di questi ultimi a partire dalle entità contenute nei vari oggetti di tipo Level. I livelli vengono popolati leggendo i file .txt corrispondenti tramite la classe LevelFileReader e vengono invece creati grazie alla classe LevelListReader. Queste due classi sono inoltre usate nella modalità di gioco alternativa che permette ai giocatori di caricare livelli custom con mappe, oggetti e mob forniti dal giocatore.

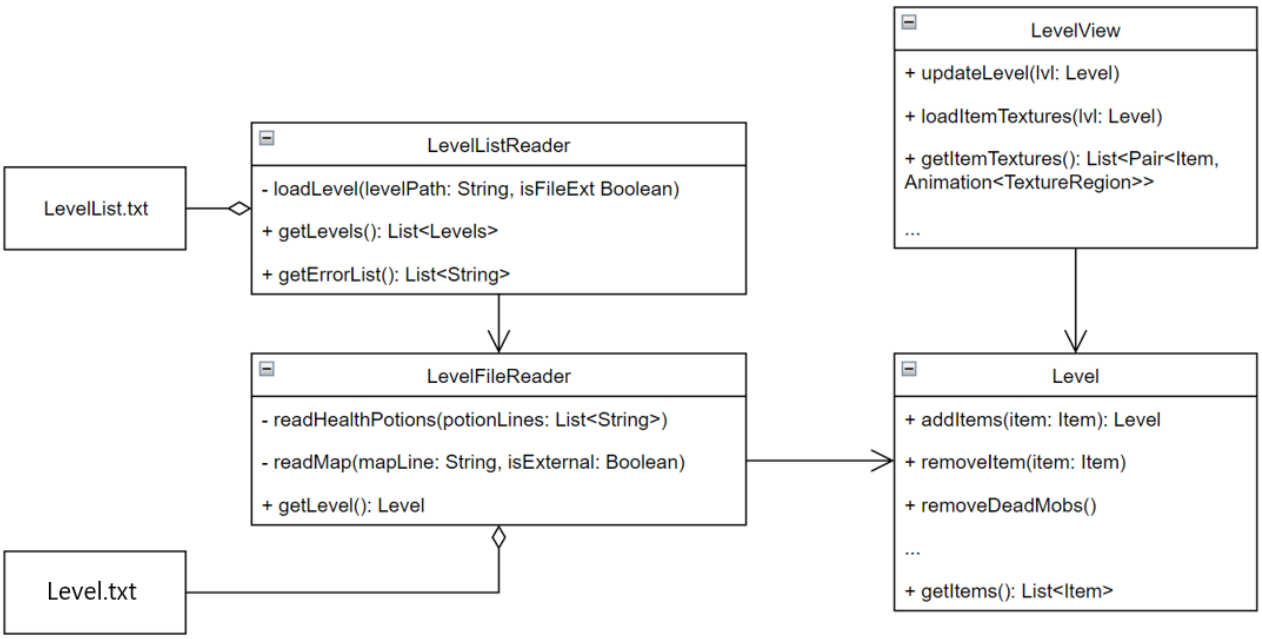


Figura 2.11 Schema UML del level, levelView e dei reader

Anche le classi sui personaggi usano il pattern Template Method, in particolare alcuni metodi della classe Character ha metodi che in futuri sviluppi possono essere implementati in Mob, come l'utilizzo di pozioni e il cambio arma. L'UML mostra i metodi che permettono ai personaggi di svolgere le azioni principali per superare i livelli (muoversi, attaccare, usare e raccogliere oggetti) e i metodi per caricare texture e il rettangolo usato per le collisioni.

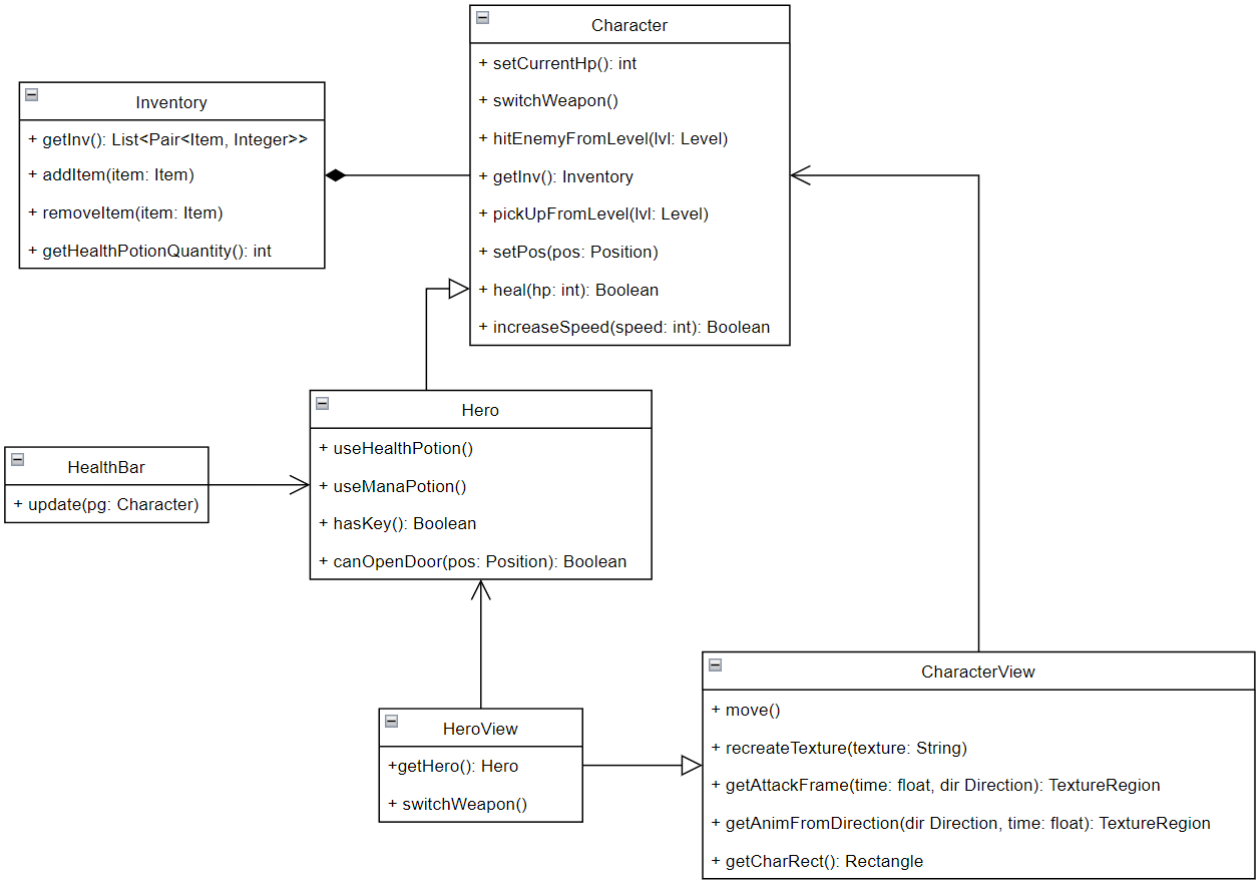


Figura 2.12 Schema UML dell'hero

Capitolo 3

Sviluppo

3.1 Test automatizzati

Abbiamo realizzato test automatici basati sulla suite di JUnit, e si trovano nel package com.unibo.tests.

Sono stati creati dei test per l'esperienza, leveling e uso delle abilità dell'eroe, per gli oggetti, dei livelli, .

3.2 Metodologia di lavoro

La parte di analisi è stata effettuata a distanza via chat vocale qualche giorno prima dello sviluppo.

Ci siamo divisi i lavori come segue:

- **Corrado stortini:** Statistiche e classe dei mob e KeyBindings. Ha deciso di implementare anche un sistema di esperienza e delle abilità per il personaggio principale.
- **Francesco Carlucci:** Vari menu tra cui quello principale, gestione dei livelli di gioco. Ha deciso di implementare le pozioni di mana e gli oggetti indossabili.
- **Jonathan Lupini:** Gestione completa dell'audio del gioco, IA dei mob (pathfinding), interfaccia delle mappe e parte della loro implementazione.
- **Lorenzo Todisco:** Implementazione di character, eroe, items (tutti tranne wearable e mana potion), menù di pausa, parte grafica del gioco (classe GameScreen) e classi per caricare livelli dai file.

Abbiamo deciso di usare il DVCS Git cercando di lavorare su branch separati in base alla feature da svolgere. Quando una feature si riteneva completata, o ad un punto funzionale, si effettuava un merge sul branch main. Eventualmente ci sono stati anche dei commit minori direttamente sul branch main (per esempio typo o javadoc).

3.3 Note di sviluppo

Corrado Stortini

Gli aspetti avanzati che ho utilizzato sono:

- Lambda expressions;

- Optional;
- Libreria libGDX;

LibGDX l'ho utilizzata nella classe di InputHandler per segnalare quando un tasto specifico viene premuto.

Su internet non c'è un algoritmo specifico per implementare il level up come lo volevo io, perciò l'ho sviluppato da zero. L'algoritmo si occupa di aumentare il livello dell'eroe di uno, portare l'esperienza dell'eroe ad una quantità sensata (se, per esempio, il personaggio aumenta di livello con 50 di esperienza, e ne riceve 60, l'eroe si troverà di livello aumentato con 10 di esperienza), aumenta la quantità di exp (esperienza) necessaria a salire di livello, aumenta vita e mana massimo, ricaricandole, e se l'esperienza rimasta dopo il level up permette al personaggio di livellare di nuovo, richiama sé stessa. Una volta raggiunto il livello massimo, l'eroe non livellerà più.

Francesco Carlucci

Ho utilizzato le seguenti features avanzate del linguaggio:

- Optional e Stream
- Lambda expressions
- Libreria esterna LibGDX

L'utilizzo della libreria ha influenzato particolarmente l'implementazione dei menu di gioco e la classe GameScreen che funge da controller dell'applicazione.

Jonathan Lupini

Non ho trovato opportunità per usare aspetti avanzati del linguaggio se non l'uso della libreria LibGdx e del tool Tiled. Ho scelto questo framework durante la fase di ricerca in quanto particolarmente adatto ai giochi 2d in stile top down e per la sua estrema portabilità.
Con Tiled abbiamo creato le mappe e tramite LibGdx abbiamo potuto estrarre dal file i dati necessari per arricchire il mondo con collisioni e zone speciali.
Ho anche fatto largo uso della libreria audio di LibGdx per implementare il mio audioManager, aggirando i limiti della libreria dettati dalla portabilità verso android (non si può controllare se un suono sia ancora in corso ad esempio).

Lorenzo Todisco

Ho realizzato le sezioni assegnatemi sfruttando il più possibile LibGDX, soprattutto per quanto riguarda la parte di View. Ho usato spesso feature avanzate di Java (Stream e Lambda e qualche Optional) per la gestione di oggetti, inventario e comandi. Non ho propriamente sfruttato il parsing di XML ma ho fatto un parsing

“custom” per caricare i livelli e il contenuto di ogni livello a partire da file .txt e una struttura di cartelle ben definita.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Corrado Stortini

Sono abbastanza soddisfatto del lavoro svolto, anche se avrei preferito creare più interfacce per poi implementarle, mentre la fase di progettazione è stata presa secondo me, troppo alla leggera. Mi sarebbe piaciuto poter sviluppare anche altre feature come degli attacchi a distanza, ma in generale il lavoro svolto mi è servito così che in futuro io possa avere qualche spunto per creare videogiochi; infatti, non penso che questo progetto possa essere una base solida, ma ci sono secondo me, degli spicchi di “genialità” da prendere in considerazione, come il Command pattern da me implementato o l’algoritmo della AI dei mob. Per quanto riguarda la mia parte, ho cercato di fare del mio meglio per essere di aiuto ai miei compagni e a lavorare sulla mia parte nel modo migliore possibile, sebbene sono consapevole che alcune cose (come i Mob) fossero strettamente legate al lavoro di altri (Mob che estende Character, creata da Lorenzo Todisco), per cui, in quei casi, il lavoro di progettazione e sviluppo è stato minore rispetto al KeyBinding o al sistema di leveling, che nonostante dipendesse anch’esso dal lavoro di Lorenzo, è stato necessario un lungo lavoro di progettazione e sviluppo.

Francesco Carlucci

Il lavoro da me svolto non mi soddisfa appieno, nonostante abbia cercato il più possibile di seguire le norme di buona programmazione insegnateci durante il corso. L’insoddisfazione, però, è causata in maggior parte dal fatto che sono stato privato di alcune sezioni che mi spettavano (sistema di combattimento), di conseguenza avrei voluto fare di più a livello di programmazione vera e propria. D’altra parte, le implementazioni da me prodotte non mi dispiacciono e sono contento di aver usufruito di un pattern noto per risolvere un problema comune.

Per quanto riguarda il mio ruolo all’interno del gruppo penso di essermi impegnato molto per le sezioni da me prodotte e sono sempre venuto in contro alle esigenze del gruppo.

Jonathan Lupini

È stato un progetto molto ambizioso a mio parere e l'averlo portato a termine per me conta indubbiamente come achievement. Non posso dirmi troppo contento della mia esecuzione, sia a livello di codice che a livello di organizzazione col gruppo.

Le classi che ho fatto sono funzionali e dispongono di riutilizzabilità ma in molti punti c'è del codice ripetuto che sicuramente avrei potuto mettere in altri metodi, la mancanza di un controller ha reso tutto molto più difficile e tornando indietro abbandonerei l'idea di animare gli sprite per aderire completamente al modello MVC.
A livello di organizzazione farei molto diversamente. Prima del progetto pensavo che aderendo ai principi di buona programmazione potessimo sviluppare feature indipendentemente e unirle a lavoro finito. Magari se si fosse un programmatore perfetto questo sarebbe vero ma parlandosi sporadicamente e senza entrare nei dettagli si arriva a design incongruenti, hack e workaround che fanno salire la difficoltà di sviluppo esponenzialmente ogni volta che si usa una scorciatoia e portano a mille redesign. Delle 80 ore per il progetto invece che una divisone equa tra coding e progettazione sono arrivato a quasi 55 ore di coding per via di redesign fatti per accomodare cose impreviste. Se dovessi rifare il progetto cercherei di adottare un modello SCRUM con meeting almeno settimanali per assicurarsi di star sviluppando tutti nella stessa direzione, possibilmente un sistema di ticket per fare in modo che ciascuno possa essere il piu possibile responsabile del suo codice (perché è capitato che ci fosse confusione su chi deve fare cosa e come), di sicuro fare almeno un'altra analisi dei requisiti a metà progetto perché quella iniziale per noi è risultata molto insufficiente a livello di dettagli e mettere delle deadline categoriche per l'implementazione di feature perché altrimenti si finisce per completare qualcosa all'80% e scoprire dopo che va rifatto.
Considerando che questo per me è stato il primo progetto di gruppo, il primo progetto con un version control system, il primo progetto con librerie non viste in classe e soprattutto il primo progetto con un monte ore definito da non superare (la parte per me più difficile) posso dirmi estremamente soddisfatto dell'esperienza acquisita e non troppo del risultato.

Lorenzo Todisco

Il codice da me realizzato è facilmente estendibile per la maggior parte tuttavia sono conscio del fatto che si potessero trovare soluzioni migliori per quanto riguarda il design (soprattutto per la view) e che il model potrebbe essere migliorato. Nonostante questo, ho cercato di usare le best practices sfruttando le capacità di LibGDX per quanto riguarda view e controller.

All’interno del team ho dato un importante contributo per le fondamenta dell’applicazione in quanto model e view dei personaggi e degli oggetti e la loro interazione nel mondo di gioco (anche attraverso la mappa, di cui ho migliorato le collisioni) sono state fatte da me. Ho inoltre facilitato la creazione di livelli creando classi per caricare i livelli in modo dinamico sia dall’interno del package dell’applicazione che dall’esterno.

Ho fornito spesso aiuto ai membri del team per risolvere bug o problemi particolarmente difficili per questioni di model ma soprattutto per view visto che sono stato un po’ più di tempo a testare LibGDX.

Penso che il progetto possa essere esteso con varie migliorie vista la facile estendibilità di mob, armi e mappe e sono in generale soddisfatto del mio lavoro, soprattutto per la parte di caricamento di livelli esterni; Tuttavia, vorrei aver sviluppato un sistema di caricamento di asset dall’esterno per facilitare la customizzazione del gioco ma penso che lo farò in futuro.

4.2 Difficoltà incontrate e commenti per i docenti

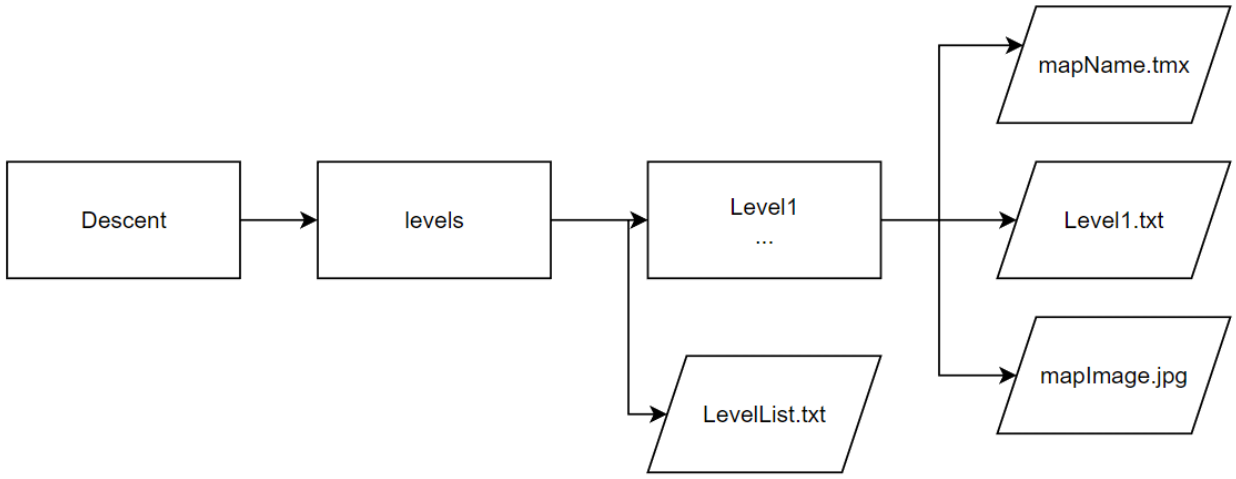
Francesco Carlucci

Sono molto contento di aver svolto un progetto così impegnativo perché penso sia molto utile per mettere in pratica ciò che è stato fatto durante il corso e per imparare a lavorare in un gruppo. L’unica nota negativa è legata proprio al lavorare in gruppo poiché non avendo fatto nessun’altro progetto in team ci si ritrova catapultati in una nuova esperienza che può creare alcuni disagi. Il mio consiglio per il futuro del corso è quello di consegnare i lavori dei laboratori settimanali in gruppo in modo che ci si possa preparare a lavorare con più persone, ciò non sarebbe comunque sufficiente a risolvere del tutto il problema, ma potrebbe giovare ugualmente.

Appendice A

Guida utente

All'avvio dell'applicazione verrà mostrato il menu principale, da cui si può iniziare una nuova partita con dei livelli di default tramite il tasto "Play". Con il pulsante "Settings" si accede alle impostazioni e con il pulsante "Quit" si esce dal gioco. C'è anche la possibilità di giocare uno o più livelli custom tramite il tasto custom levels: per farlo, bisogna prima andare nella cartella AppData/Roaming/Descent e aggiungere una serie di cartelle come mostrato in figura.



I rettangoli rappresentano le cartelle, i parallelogrammi i file

Il file LevelX.txt (dove X può essere un qualunque numero) deve essere formattato secondo questa linea guida:

```
//Format:
//EnumName ITEM_NAME xCoordinate,yCoordinate
//MapImpl path xSpawnCoordinate,ySpawnCoordinate unitScale

MapImpl levels/Level2/Interior.tmx 800,272 0.275
audio/music/Danmachi.mp3
HealthPotionStats BASIC_HEALTH_POTION 1406,830
HealthPotionStats MEDIUM_HEALTH_POTION 1459,950
ManaPotion 0.25 800,372
Wearable HealthRing Hp:0.25 800,272
DoorKey 215,815
DoorPosition 798,638
MobStats TROLL 5 470,700
WeaponStats GREATAXE 1359,830
```

Nel file mapName.tmx invece sono presenti i tile associati alla mappa mapImage.jpg.

Una volta iniziata la partita, il giocatore può muoversi usando le frecce “sopra, sotto, destra, sinistra”. Per attaccare si deve usare la barra spaziatrice, per raccogliere gli oggetti il tasto “E”. Per usare le pozioni del mana si usa il tasto “F”, e quelle del mana si usano con il tasto “M”. Per aprire la porta e andare al livello successivo bisogna premere il tasto “K” quando si è sulla porta con la chiave nell’inventario. Per aprire il menu delle abilità bisogna premere il tasto “T”, dove si possono vedere tutte le abilità, il livello a cui si possono usare e il tasto con cui usarle. Si può premere il pulsante “Esc” mentre si è in partita per aprire il menu di pausa, da cui si può uscire dal gioco o tornare al menu principale. Le barre in alto a destra sono per la vita (quella verde) e per il mana (quella blu). La barra bianca in basso allo schermo è la barra dell’esperienza e una volta riempita l’eroe sale di livello e recupera tutta la vita e il mana (oltre che ad aumentare la quantità massima di entrambi). Gli oggetti che si trovano in giro aiutano l’eroe in modi diversi, per esempio le armi aumentano il danno dell’eroe, le pozioni possono essere usate per recuperare vita o aumentare il mana massimo, e gli accessori o aumentano la vita, o aumentano il danno dell’arma o danno esperienza, o una qualsiasi combinazione delle tre.

Appendice B

Esercitazioni in laboratorio

Corrado Stortini

Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=87880#p135441

Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p136569

Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p137703

Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p138797

Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p139934

Francesco Carlucci

- Lab 7: https://virtuale.unibo.it/mod/forum/discuss.php?d=88829#p142328
- Lab 8: https://virtuale.unibo.it/mod/forum/discuss.php?d=89272#p142321
- Lab 9: https://virtuale.unibo.it/mod/forum/discuss.php?d=90125#p142322
- Lab 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=91128#p142323