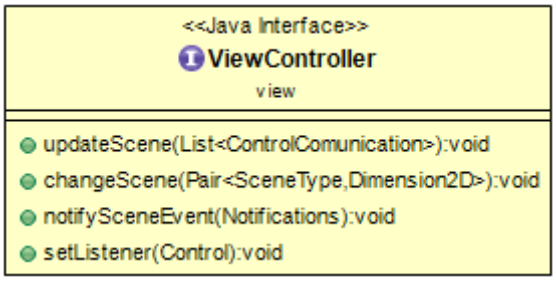
DESIGN DETTAGLIATO - VIEW

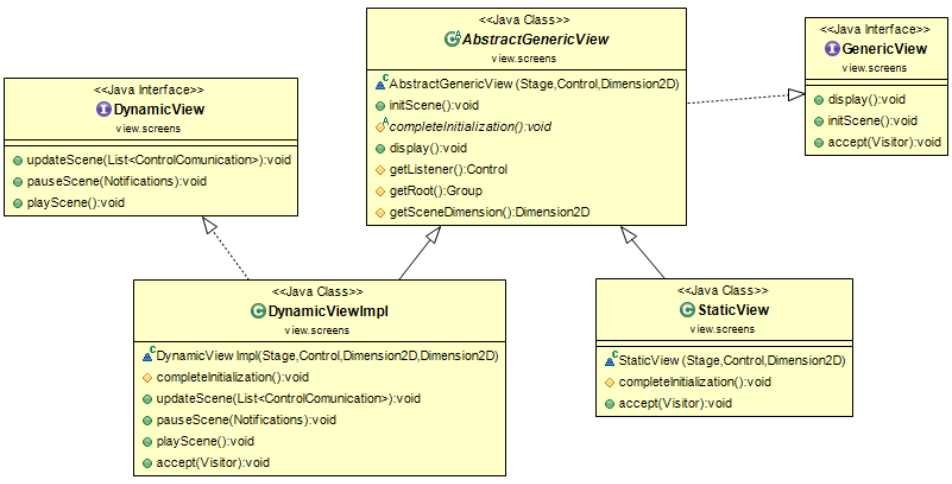
La View espone al controller una singola classe per l'interfacciamento, in modo che un aggiornamento nell'architettura interna progettata non comporti delle modifiche anche nel resto dei moduli dell'applicazione.



Le funzionalità offerte da questa interfaccia sono:

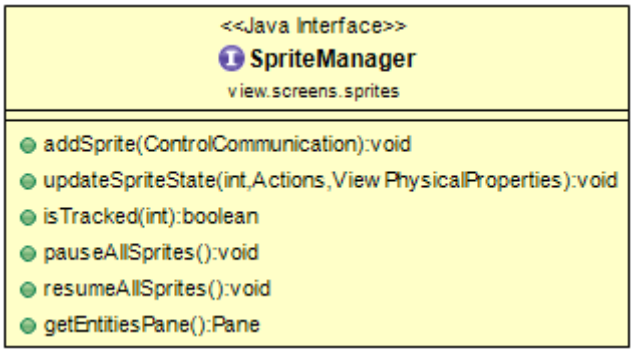
- La creazione e la visualizzazione di una nuova schermata, la cui tipologia è decretata da un elemento di un enumerazione che definisce i tipi di scena disegnabili da questo modulo;
- L'aggiornamento della schermata creata, se questa supporta l'aggiornamento;
- La notifica alla schermata creata di un evento particolare, quali per esempio lo stato di vittoria o di sconfitta, definiti grazie ad un enumerazione;
- La definizione di un oggetto fornito dal controller a cui questo modulo potrà fare eventuali richieste di informazioni o tramite cui inviare le propie.

La prima difficoltà sorta nella realizzazione di queste funzionalità è stato la creazione di un meccanismo di cambio della schermata che permettesse la creazione di scene con caratteristiche e proprietà anche molto diverse tra loro, come quelle che possono avere un menù statico e una schermata di gioco.
Perché potessero essere gestite in modo simile dal ViewController le diverse scene sono state sviluppate a partire da una stessa interfaccia generica, la cui implementazione si va a specializzare grazie all'uso del Template Method pattern applicato al momento della loro inizializzazione.



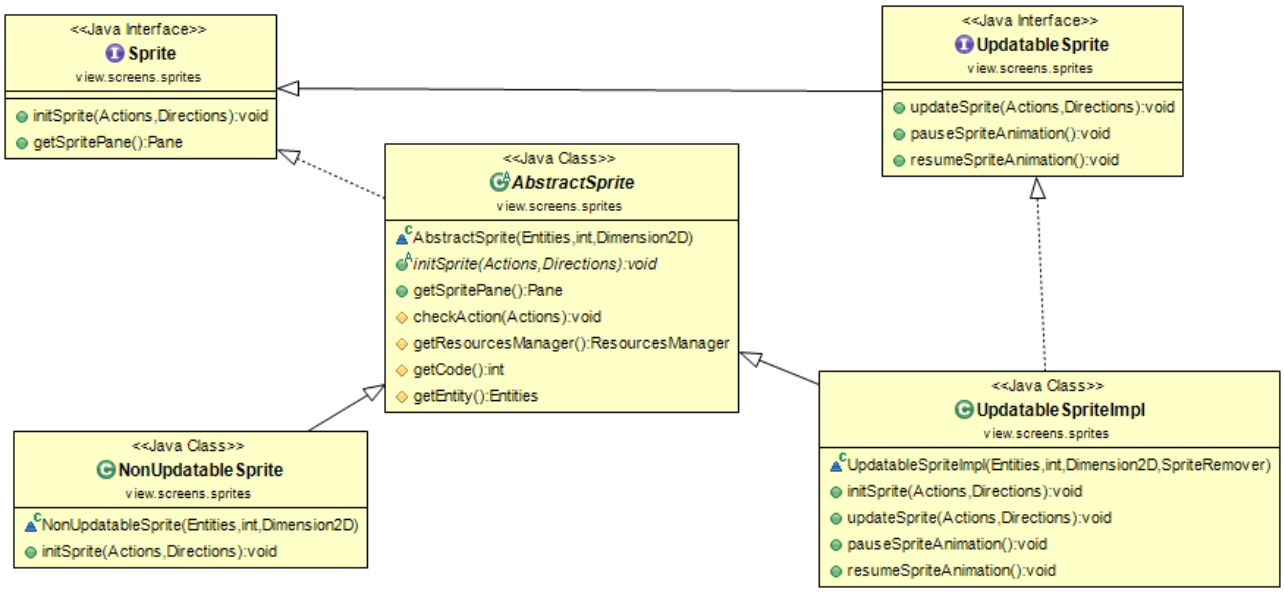
Questo ha permesso di utilizzare al momento della creazione delle scene il Factory pattern, garantendo così un maggior grado di pulizia nel codice e la libertà futura di aggiungere altre tipologie di scene senza modificare il funzionamento dell'architettura creata.
Le schermate aggiornabili però, necessitando di metodi aggiuntivi per la loro gestione, vanno a ereditare da una seconda interfaccia che ne definisce le ulteriori funzionalità. Per utilizzare questi metodi si sarebbe reso necessario un cast dall' interfaccia generica a quella più specifica. Per ovviare a questo problema ottenendo un riconoscimento del tipo dinamico delle scene istanziate dalla factory, e quindi avere un comportamento diverso in base a questo, ci si è avvalsi del pattern Visitor.
Un altro metodo molto più semplice per ovviare al problema sarebbe stato quello di eliminare completamente la factory e istanziare direttamente le scene con l'interfaccia più opportuna, considerato comunque che allo stato attuale ne sono presenti solamente due tipologie. La decisione di avvalersi comunque di questi pattern, e quindi articolare ulteriormente un'architettura che sarebbe stata molto più semplice, è stata presa pensando a una possibile, e molto probabile, espansione futura, quando i tipi di schermata gestiti da questo modulo saranno molti di più. Un altro possibile tipo di schermata per esempio sarebbe potuta essere una per la visualizzazione di filmati, utilizzabili come intermezzo fra i vari livelli.
La metodologia adottata invece permette di aggiungere qualsiasi tipo di schermata, sempre sotto l'interfaccia GenericView, senza andare a modificare la ViewControllerImpl che le gestisce.

Definita questa parte, la sezione di questo modulo che ha richiesto una maggiore complessità architetturale è stata quella che gestisce la rappresentazione del mondo di gioco e quindi l'aggiornamento delle entità che lo compongono e delle relative animazioni.
La movimentazione e la gestione delle entità del gioco è delegata dalla schermata di tipo dinamico a una seconda classe, lo SpriteManager.



Questa classe possiede al suo interno una lista degli sprite da visualizzare a schermo, a cui è possibile in qualsiasi momento aggiungere ulteriori elementi, e si occupa di gestire l'aggiornamento della posizione e dell'animazione corrente, entrambe basate sulle informazioni inviate dal controller. Basandosi sull'analisi di particolari azioni dell'entità, come quella di morte, si occupa anche della rimozione di queste dallo schermo.
Anche le classi che mantengono le informazioni sugli sprite attivi sono state create basandosi sui

principi di ereditarietà e polimorfismo, ottenendo così una diversificazione fra le tipologie di figure potenzialmente visualizzabili.

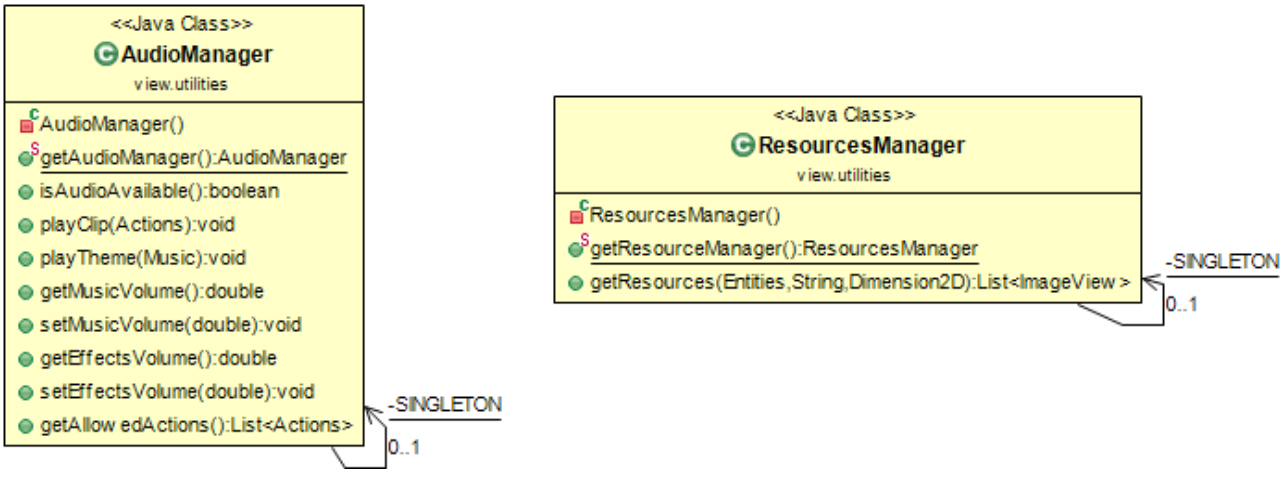


La distinzione è stata fatta principalmente tra sprite animati e non, aggiungendo ai primi anche le funzionalità per l'aggiornamento dell'animazione. In questo modo si sono ottenuti due tipologie di soggetti che possono popolare il mondo di gioco :

-Statiche, che sono usate per rappresentare le parti invariabili del mondo, quali possono essere i muri e le piattaforme;

-Dinamiche, usate per gli elementi attivi del mondo di gioco, come personaggi e mostri.

La gestione delle immagini è stata affidata a una singola classe, il ResourceManager, creata utilizzando il Singleton pattern in modo da avere una singola istanza possibile durante l'esecuzione dell'applicazione, e così poter utilizzare una sorta di meccanismo di caching delle risorse, riducendo gli accessi in lettura al disco.
Una struttura analoga è stata usata anche per l'AudioManager, incaricato di gestire le musiche di gioco e gli effetti sonori.



Per quanto riguarda la divisione in package, è stata fatta cercando di tenere divise per ambiti di funzionamento le diverse classi e quindi ottenere un accesso veloce ai diversi moduli che compongono la struttura. Si è cercato anche di ridurre la visibilità delle classi non necessarie al controller in modo da avere due domini completamente separati.