jSokoban v1.0
Report

Sokoban in Java

Progetto del corso

Programmazione ad Oggetti

Submitted by
657780 - Ferretti Davide

Date:
September 24, 2015

Contents

1 Analisi

Scopo dell'applicazione è realizzare un applicativo che cloni il gioco sokoban. L'applicativo dovrà rispondere ai seguenti requisiti:

Modalità storia
: Permettere all'utente di giocare ad una serie di livelli preesistenti.

Editor di livelli
: Possibilità di accedere ad una modalità che permetta la creazione e il salvataggio di quest'ultimi.

Livelli personalizzati
: Possibilità di caricare e quindi giocare livelli personalizzati creati tramite l'editor.

Controlli da tastiera
: Leggere correttamente gli input da tastiera dati dall'utente nelle fasi di gioco e editing.

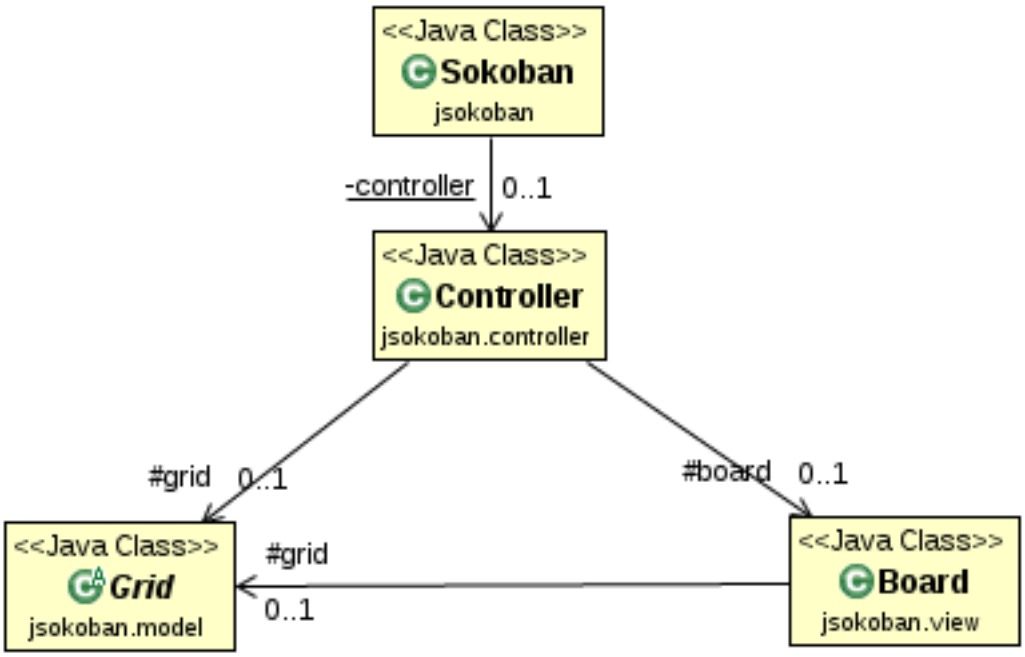
Interfaccia grafica
: Eseguire una corretta visualizzazione delle varie fasi.

Dai requisti risulta necessario lo sviluppo di un'applicazione basata su una GUI che permetta all'utente di passare dal gioco all'editor in maniera rapida. Si opererà poi all'interno dell'applicazione attraverso scorciatoie sulla tastiera. Si vorrano anche gestire in maniera opportuna e separata i due diversi utilizzi.

2 Progettazione e organizzazione in package

L'intera progettazione ha seguito come linea guida il pattern Model-View-Controller, grazie al quale si è riuscita a creare una suddivisione del codice che permette il riuso e l'estensione dei vari aspetti descritti all'interno dell'applicazione.

Nello specifico avremo il seguente schema UML di base che rappresenta in generale il funzionamento dell'applicativo:



2.1 Pattern utilizzati

⊙ **Strategy Pattern**

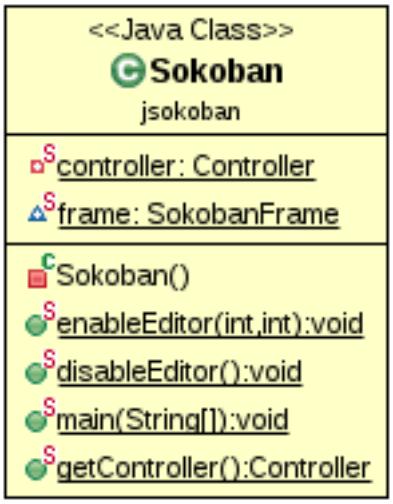
PlayerGrid implementa il pattern Strategy attraverso il metodo `setLevel()` il quale a secondo del livello caricato imposta alcuni parametri in modo che il controller permettà di eseguire azioni prima non disponibili.

2.2 Organizzazione dei package

Il progetto è composto da diversi package i quali contengono al loro interno una o più classi. Qui di seguito una vista più dettagliata con relative descrizioni.

2.2.1 Package `sokoban`

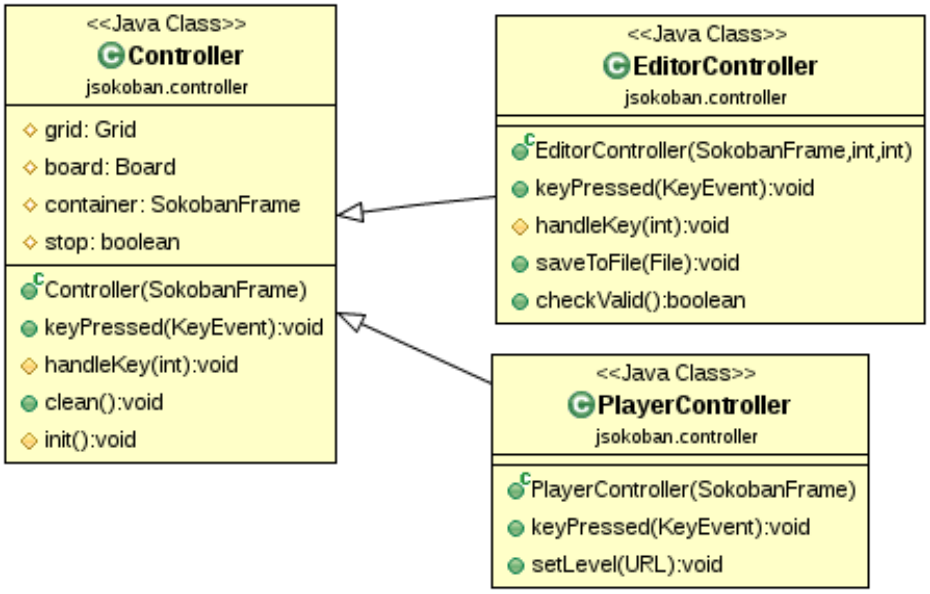
Il package principale che contiene solo la classe `Sokoban` dalla quale l'applicazione viene instanziata.



Sokoban
: Questa classe rapprensenta il punto di accesso all'applicazione. Crea e mantiene l'unica instanza di `Controller` di tutta l'applicazione. Difatto espone i metodi `enableEditor()` e `disableEditor()` che hanno la funzione di modificare, rispettivamente, il controller in `EditorController` oppure in `PlayerController`.

2.2.2 Package `sokoban.controller`

Questo package contiene la classe `Controller` assieme alle sue specializzazioni: `EditorController` e `PlayerController`.



Controller

Questa classe estende `KeyListener`, crea al suo interno un'istanza di `Grid` che verrà utilizzata come Model seguendo il pattern MVC e un'istanza di `Board` da usare invece come View. Il suo metodo `clean()` permette di rimuoversi dai keylistener del frame che si sta ascoltando e rimuovere board.

PlayerController

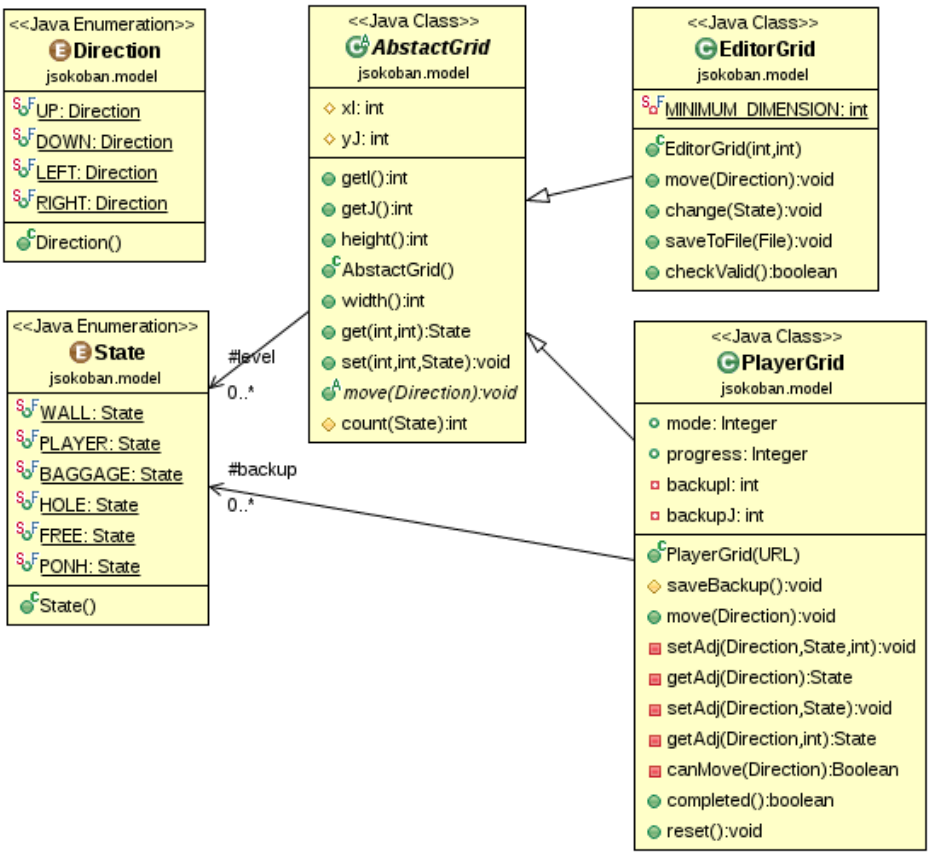
Estensione di Controller che registra la Grid come `PlayerGrid`. Aggiunge funzionalita a `keyPressed()`. Inoltre il metodo `setLevel()` permette di cambiare livello utilizzato.

EditorController

Estensione di Controller che registra la Grid come `EditorGrid`. Aggiunge funzionalita a `keyPressed()`. Permette inoltre di accedere ai metodi `saveToFile()` e `checkValid()` dell'`EditorGrid`.

2.2.3 Package `sokoban.model`

Questo package contiene la classe astratta `AbstractGrid` e le sue due implementazioni `PlayerGrid` e `EditorGrid`, oltre a loro contiene le definizioni degli enum `State` e `Direction`.



State

Enum che difinisce i vari stati possibili per ogni cella della griglia.

Direction

Enum che definisce le diverse direzioni che si possono prendere.

AbstractGrid

Classe astratta che fornisce una griglia `level[][]` che contiene il livello corrente correlata dei metodi per accedervi e modificarla, le coordinate dell'oggetto principale (il quale potrebbe essere il puntatore dell'editor o il giocatore) e le dimensioni della griglia. Definisce inoltre il metodo astratto `move()` che verrà poi implementato nelle specializzazioni di `AbstractGrid`. Il vantaggio di questa classe astratta è quello di poter aggiungere nuove logiche di gioco in modo semplice.

EditorGrid

Estende Grid ereditandone i metodi e le proprietà. Implementa il metodo `move()` in modo tale da rispettare le regole che si sono volute dare all'Editor.

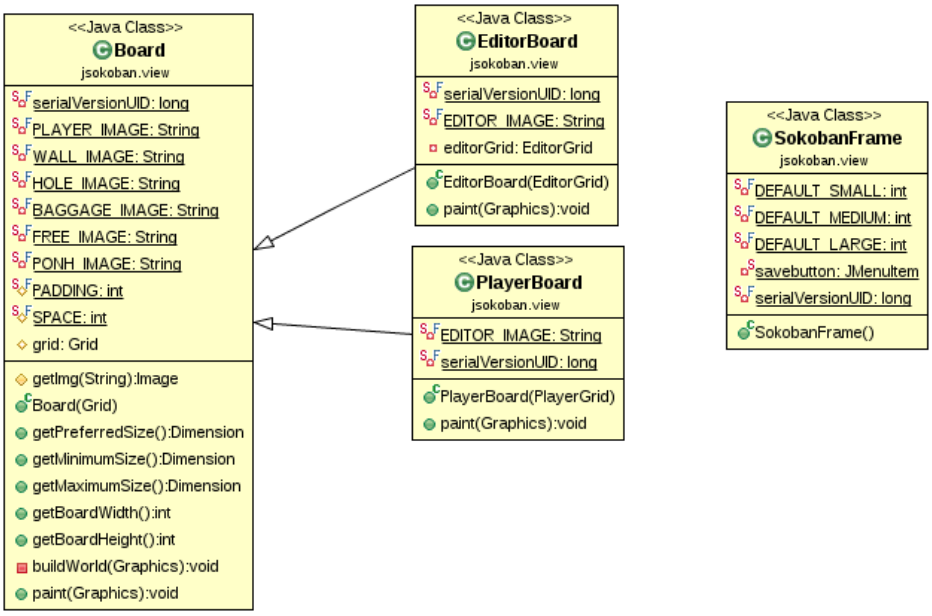
Altri metodi implementati sono `saveToFile()`, utilizzato per salvare il livello attuale su file; `checkValid()`, per controllare che il livello corrente sia valido; `change()`, usato per cambiare la disposizione degli elementi nella griglia.

PlayerGrid

Estende Grid, implementa il metodo `move()` in modo da rispettare la logica del gioco, inoltre gestisce una copia di backup in modo tale da permettere il reset del livello. Fornisce un metodo `completed()` che restituisce un booleano che indica se la partita è terminato o meno. In futuro potrebbe essere estesa per aggiungere o modificare la logica del movimento. In particolare la seguente classe implementa uno scanner di file che controlla l'input, verificando che esso sia corretto.

2.2.4 Package `sokoban.view`

Questo package contiene le varie classi che realizzano la visualizzazione dell'ambiente, nello specifico contiene `SokobanFrame` che rappresenta il frame dell'applicazione e `Board` con le sue specializzazioni che disegna il campo.



SokobanFrame

Questa classe rappresenta il frame principale dell'applicazione. Crea al suo interno i vari menu dell'applicazione e richiama il controller per l'esecuzione delle varie operazioni.

Board

Classe che estende JPanel. Permette di disegnare il "mondo" di gioco, sia che ci si trovi in modalità editor o meno, per un dato model. Implementa al suo interno il metodo `buildWorld()` che viene chiamato ogni volta da `paint()` ereditato da JPanel.

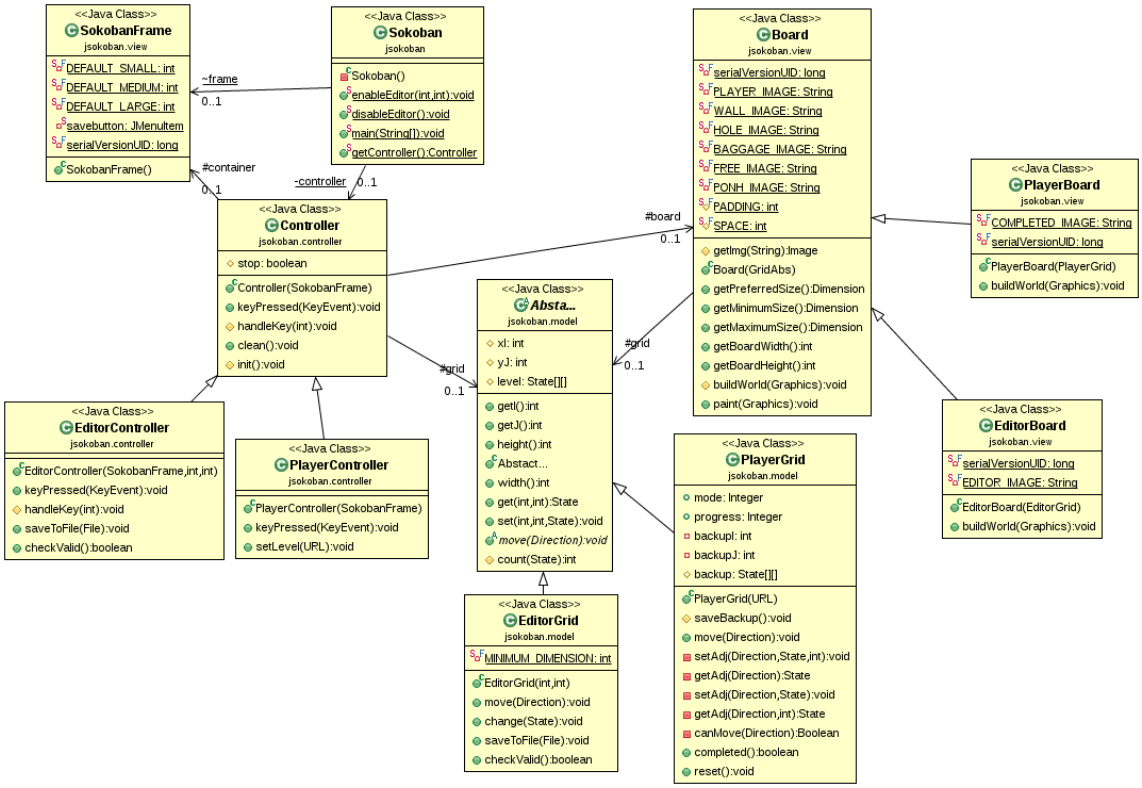
EditorBoard

Estende Board aggiungendo del comportamento a `buildWorld()` relativo all'editor. Nello specifico aggiunge la visualizzazione di un cursore di modifica.

PlayerBoard

Estende Board aggiungendo del comportamento a `buildWorld()` relativo all'ambiente di gioco. In particolare, a fronte del completamento di un livello, segnala graficamente la vittoria.

2.2.5 Uno sguardo all'insieme



Come si puo vedere dallo schema complessivo:

- Le specifiche del pattern MVC sono state rispettate.
- Si nota anche che l'unico controller di tutta l'applicazione viene creato dal main, che si occuperà poi anche di alternare l'istanza puntata in modo che sia un `EditorController` ed un `PlayerController` in base alle necessità.
- Si puo anche vedere come le diverse specializzazioni delle classi principali quali `Controller`, `Board` e `Grid` non vengano mai direttamente create bensì vengano passate a variabili che hanno come tipo la classe che estendono.

3 Implementazione

Una volta conclusa la fase di progettazione è cominciata l'effettiva realizzazione dell'applicativo. In questa fase sono state eseguite delle scelte che ne hanno determinato poi la reale funzione di alcuni metodi.

3.1 File dei livelli

Ogni livello del gioco è salvato in un file che in corso d'opera è stato modificato più volte a livello di struttura. La struttura finale risulta essere:

Riga 1
: Contiene 2 valori numeri che rappresentano rispettivamente la modalità a cui afferiscono e il loro successivo(!=0 solo in modalità storia).

Riga 2
: Contiene 2 valori numeri che rappresentano l'altezza e la larghezza della griglia che verra creata

Dalla riga 3 in poi
: Sono i caratteri che permetterano l'inizializzazione della griglia con i giusti stati.

Come esempio, vediamo la struttura di `story1.txt`:

```
1 2
9 9
WWWWWWWWW
WW...WWWW
WW...W..W
WWW....HW
WWW.WWWHW
W.B.WWWHW
W.BBWWWWW
WP..WWWWW
WWWWWWWWW
```

3.2 Logica del gioco

Nel gioco Sokoban originali il giocatore ha l'obiettivo di spostare alcune scatole disposte all'interno del livello in alcuni punti fissati.

Nella mia versione questa logica non è stato implementata del tutto uguale. Infatti mentre nell'originale la scatola una volta messa in uno dei punti di arrivo puo essere ancora spostata. Ciò non avviene nella versione da me implementata dal momento che i punti di arrivo sono rappresentati mediante dei buchi che vengono chiusi in modo definitivo.

Questo non intacca però la fruibilità del gioco infatti tutti i livelli della versione originale sono completabili in questa versione.

4 Conclusioni

Sviluppo

Lo sviluppo del progetto è stato lineare, unici cambiamenti in corso d'opera hanno riguardato solo la rifattorizzazione del codice anche se di un certo spessore. Per quanto riguarda i tempi dovrei trovarmi sulle 100 ore.

Criticità riscontrate

Unico vero problema durante lo sviluppo è stata la gestione dei controller i quali in un primo momento venivano creati ogni volta che si cambiava modalità azzerando il precedente. Ciò però portava ad una non completa sparizione dei controller precedenti che a volte continuavano a funzionare, si è passati quindi al controller unico nel main che al suo interno genera il resto. Se in un primo momento questa soluzione andava bene alla fine della refattorizzazione sono venuti fuori altri problemi relativi questa volta alle istanze del controller a cui il frame mandava gli eventi, si è passato quindi ad utilizzare, all'interno dell'applicativo, l'unico controller creato nella main.

Mancanze

All'interno della proposta di progetto era stata indicata la possibilità di utilizzare all'interno del gioco caselle speciali. L'intera applicazione e stata progetta in maniera da rendere possibile questa mia proposta, l'implementazione effettiva di queste caselle manca per mancanza di tempo ma grazie alla struttura dell'applicazione non è difficile creare o estendere una nuova griglia che implementi le logiche di gioco per queste nuove case.