Progetto Programmazione Concorrente e Distribuita
Prima Parte

Suierica Bogdan Ionut 1008089

December 24, 2014

1 Introduzione

1.1 Scopo del documento

Lo scopo del documento è quello di presentare le principali scelte architetturali del progetto.

2 Descrizione Progetto

Il progetto realizzato permette di ordinare un puzzle di caratteri. Il file in input contiene il testo rappresentato dal puzzle disordinato. Il file in output contiene il testo rappresentato dal puzzle disordinato organizzato in una riga, il puzzle ordinato in forma tabellare, e la dimensione della tabella.

3 Componenti e Classi

Tutti i campi dati delle classi utilizzate nel progetto sono privati e dotati di metodi pubblici per permettere l'accesso. Questo offre mantenibilità, flessibilità ed estensibilità del codice. Nel progetto ho cercato di dividere tutti i vari compiti in classi.
Si descrivono di seguito le componenti e le classi individuate in fase di progettazione del sistema.

3.1 Package puzzle

Per questa prima parte del progetto, puzzle rappresenta il package globale del progetto. La scelta di adottare un solo package è dato dalle classi utiliz-

zate nel sistema, in quanto tutte le classi erano relative allo stesso argomento e cioè a quello di ordinare un puzzle di stringhe da un file in input.

3.2 Interface SolverAlgorithm

L'interfaccia è stata utilizzata per permettere una futura estensibilità del codice. È quindi possibile inserire un differente algoritmo di risoluzione, presentando quindi alternative diverse allo sviluppatore e/o utilizzatore. È simile al design pattern Strategy, il quale permette di presentare diverse soluzioni di un problema. In futuro sarà quindi necessaria la sola implementazione dell'interfaccia AlgorithmSolver e del suo metodo "solve()".

3.3 Classe IOReader

Questa classe rappresenta la lettura da file del puzzle disordinato in forma tabellare. Quando avviene la lettura , calcola la dimensione della tabella attraverso dei contatori che contano i Tile con idNord="VUOTO" e idOvest="VUOTO".

3.4 Classe IOWriter

Questa classe rappresenta la scrittura su file e come detto in precedenza, scrive il testo rappresentato dal puzzle disordinato, il puzzle ordinato in forma tabellare e la dimensione della tabella.

3.5 Classe Tile

Questa classe rappresenta il singolo pezzo del puzzle. Ogni pezzo ha un id univoco e gli id dei pezzi che li stanno vicino nelle quattro direzioni.

3.6 Classe Puzzle

Questa classe rappresenta l'insieme di pezzi del puzzle. E' caratterizzata da un ArrayList di **Tile** , due interi che indicano la dimensione della puzzle, una stringa che indica il path a cui si trova il file in input e un riferimento all'oggeto **IOReader** .
Inoltre alla costruzione dell'oggetto **Puzzle** avviene la lettura da file e l'ArrayList di **Tile** viene popolato.

3.7 Classe PuzzleToSolve

La classe **PuzzleToSOlve** implementa l'interfaccia **SolverAlgorithm**. Ha il compito di salvarsi il puzzle disordinato e restituirlo ordinato attraverso il metodo realizzato "solve()", fornito dal interfaccia **SolverAlgorithm**.

4 Logica Algoritmo di ricostruzione del puzzle

Per l'algoritmo di ricostruzione uso l'interfaccia **SolverAlgorithm** in cui viene reso disponibile il metodo **solve** che accetta un **Puzzle** come parametro. Il metodo **solve** implementato nella classe **PuzzleToSolve** chiama altri due metodi : **SolveFirstCol** e **SolveRemainingTile**. Il primo metodo serve a ordinare solamente la prima colonna del puzzle. Il secondo serve a ordinare il resto del puzzle partendo dalla prima colonna. Ho scelto di fare cosi pensando alla seconda parte del progetto che tratta la concorrenza.

5 Correttezza

Precondizione : file formattato come da specifica.

5.1 Lettura da file

Il corpo del metodo "readContent" nella Classe **IOReader** è costituito da un istruzione *try* che indica che nel blocco che segue può venire sollevata un'eccezione ed infatti, se questo dovesse succedere per un formattazione non ammessa del file, provoca un salto all'istruzione *catch* in cui viene gestita l'eccezione occorsa. Dopo la lettura è vero che il **Puzzle** è riempito.

5.2 Algoritmo di ordinazione

La ricostruzione del puzzle è provocato dai due metodi della classe **PuzzleToSolve** : "solveFirstCol" e "solveRemainingTile", in cui viene creata una copia del puzzle disordinato in modo da avere a disposizione tutti gli elementi del puzzle.
Il primo metodo determina la ricostruzione della prima colonna del puzzle. La correttezza di questo metodo è data dalla condizione di uscita del ciclo *for* più esterno, cioè quando si uscirà dal ciclo si avrà riempito tutta la prima colonna. Inoltre il ciclo *for* interno che scorre il puzzle copiato disordinato contiene un'altra condizione di uscita che indica il termine della lettura quando l'elemento idoneo è stato trovato.

Il secondo metodo determina la ricostruzione del resto del puzzle. In questo caso abbiamo due cicli *for* annidati che scorrono rispettivamente righe e colonne con la differenza che il ciclo che scorre le colonne parte da 1, in quanto abbiamo già ordinato gli elementi della prima colonna. Il ciclo *for* più interno che scorre il puzzle copiato disordinato presenta un'altra condizione d'uscita che serve a non fare calcoli inutili e cioè andare a cercare anche dopo aver trovato l'elemento.
All'uscita siamo sicuri che il puzzle è stato ordinato interamente.