Informazioni sul documento

Nome documento	Relazione Seconda Parte del progetto PuzzleSolver
Versione documento	v.2.0.0
Data redazione	2015-01-10
Redattore	Tesser Paolo

Sommario

Lo scopo del documento è quello di fornire una presentazione della seconda parte del progetto PuzzleSolver da realizzare, descrivendo e motivando le scelte attuate in questa fase.

Indice

1 Cambiamenti e Aggiunte **3**
1.1 Estensioni 3
1.1.1 Package solver 3
1.2 Nuove gerarchie 3
1.2.1 Package solver 3
1.2.2 Package logger 3

2 Algoritmo di risoluzione (parallelo) **5**

3 Gestione dei Thread **6**
3.1 Numero Thread 6
3.2 Interferenze, Deadlock, Attesa attiva 6

4 Costrutti di concorrenza **7**
4.1 Oggetto condiviso 7
4.1.1 Synchronized 7
4.2 wait() 7
4.3 notifyAll() 7

1 Cambiamenti e Aggiunte

In questa sezione verranno descritti i cambiamenti apportati alla precedente versione del programma per permettere all'algoritmo di risoluzione di essere parallelizzato.
In generale non vengono effettuate particolari rivisitazioni della precedente struttura, ma vengono introdotte nuove gerarchie ed estese alcune precedentemente create.
Il client rappresentato dalla classe **PuzzleSolver** andrà modificato solo nella scelta della politica di esecuzione dell'algoritmo, che da sequenziale passerà a parallela utilizzando la nuova classe descritta nella sezione 1.1.1.

1.1 Estensioni

1.1.1 Package solver

Questo package contiene le classi che gestiscono la risoluzione del puzzle.
Viene estesa la gerarchia che comprendeva alla base l'interfaccia **SolverStrategy** e come sottotipo la classe **SolverAlgStrategy** con una nuova classe **SolverParStrategy** responsabile della risoluzione parallela del puzzle. La nuova classe avrà all'interno del metodo **executeSolve** il flusso di quali e quanti thread andranno lanciati per ordinare il puzzle.

1.2 Nuove gerarchie

1.2.1 Package solver

Questo package contiene le classi che gestiscono la risoluzione del puzzle.
In esso vengono aggiunte le classi che contengono l'attività logica che i diversi thread andranno ad eseguire.
La gerarchia creata ha alla base una classe astratta **BasicThread**, che implementa l'interfaccia **Runnable**, utilizzata per memorizzare i membri condivisi da tutti i task come il riferimento all'oggetto condiviso dai Thread per comunicare tra loro o il riferimento all'oggetto Puzzle da risolvere.
Viene estesa da **AngleTileThread**, **FirstColThread**, **RowThread**.
AngleTileThread è il task che ha il compito di cercare e ordinare, nella struttura che contiene il puzzle risolto, il pezzo avente idWest e idNorth uguali a VUOTO e quello avente e idSouth uguali a VUOTO.
FirstColThread è il task che ha il compito di ordinare la prima colonna del puzzle. Per farlo può partire dai precedenti pezzi trovati a seconda di quanto specificato nel paramentro attuale del task durante la sua creazione. I valori che potrà ricevere saranno: "up" per partire dal basso e salire fino alla metà della colonna, mentre "down" per partire dal basso e arrivare sempre fino alla metà.
RowThread è il task che serve per riordinare tutte le righe del puzzle a partire dal primo elemento di ciascuna di essa.

Tutti queste classi svolgono il loro compito nel metodo **run()**.

1.2.2 Package logger

Questo package contiene solamente una classe che mi serve per la gestione di un file testuale di log che viene usato per tracciare l'esecuzione del programma in alcuni punti.

Essendo non importante ai fini delle richieste del proponente non ne verrà fornita una rappresentazione grafica.

In seguito viene illustrato il package solver contenente sia la gerarchia ampliata degli algoritmi di risoluzione, sia quella dei Thread che vengono avviati per ordinare il puzzle.

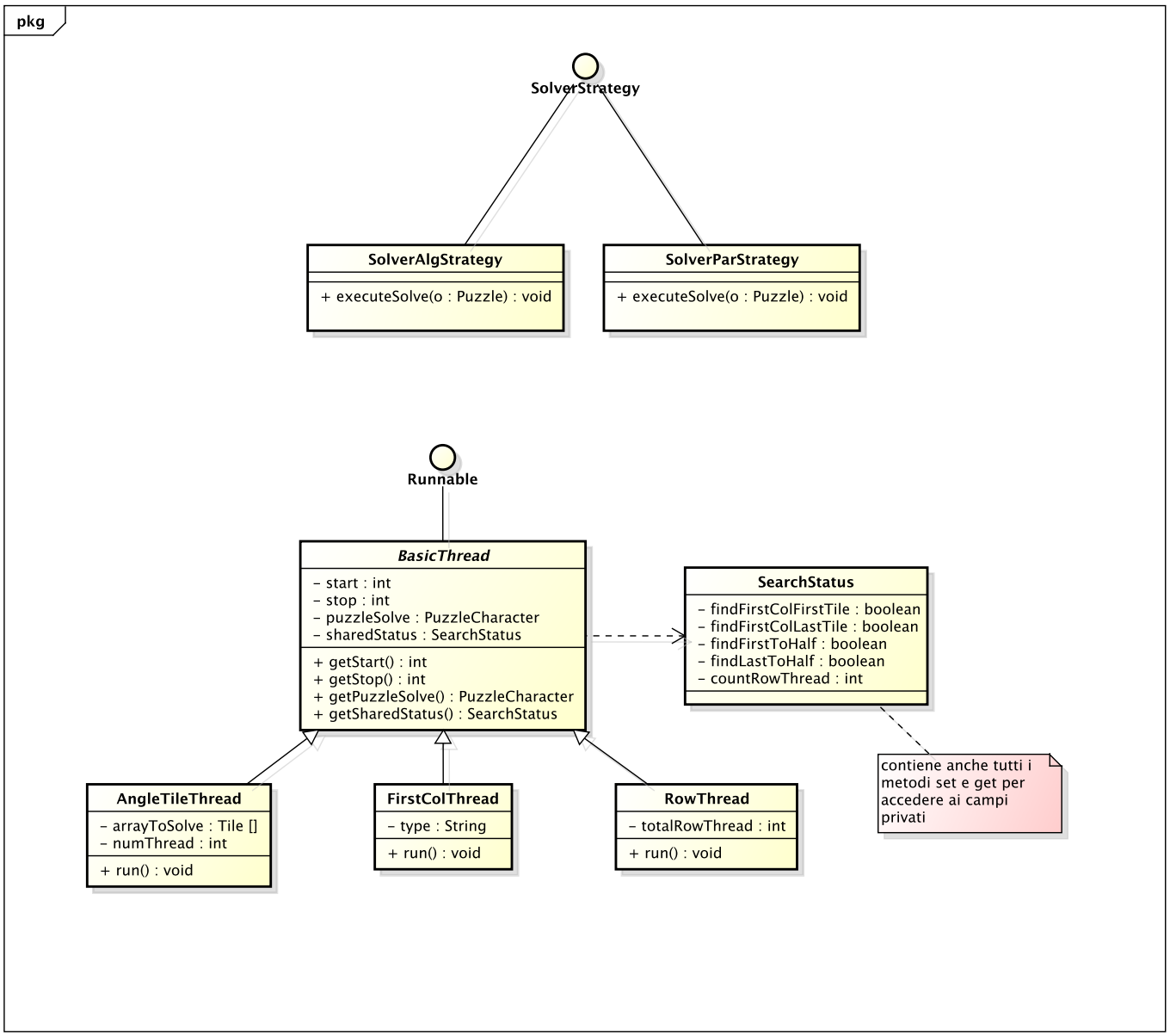

Figura 1: Package Solver

2 Algoritmo di risoluzione (parallelo)

L'algoritmo scelto per la risoluzione del puzzle è tipo di parallelo come richiesto dalla specifica relativa alla seconda parte del progetto.
Per arrivare alla soluzione, le strutture dati introdotte nella prima parte rimangono invariate. Si userà quindi sempre una HashMap per i tasselli del puzzle ancora disordinati e un array bidimensionale per i vari pezzi in ordine corretto.
Se nella precedente versione queste erano le uniche due strutture dati di cui l'algoritmo aveva bisogno, per la versione parallelizzata viene usato anche un array mono dimensionale di oggetti Tile che contiene in ordine casuale i valori della HashMap.
Di seguito vengono esposte le sequenze che vengo eseguite e i thread che vengono lanciati dall'algoritmo, correlate da grafici che mostrano come essi agiscano sulle strutture dati utilizzate.

1. **Ricerco il primo elemento del puzzle (quello in alto a sinistra) e quello in basso a sinistra.**
   TO DO;
2. **Ordino la colonna più a sinistra (quella con i tasselli aventi id ovest uguale alla stringa VUOTO).**
   TO DO;
3. **Ordino tutte le righe.**
   TO DO.

3 Gestione dei Thread

Nella seguente sezione vengono descritte le conseguenze che possono avvenire con l'avvio dei thread necessari alla risoluzione in parallelo del puzzle.

3.1 Numero Thread

Si può sempre valutare il caso peggiore e sapere al massimo quanti thread ci saranno in esecuzione nelle diverse fasi dell'esecuzione dell'algoritmo.
Di seguito verrà fornito il numero, illustrandolo in rapporto al flusso dell'algoritmo già descritto nella sezione 2.

1. Durante la ricerca e la sistemazione del primo elemento del puzzle (quello in alto a sinistra) e quello in basso a sinistra TO DO;
2. Durante l'ordinamento della colonna più a sinistra TO DO;
3. Durante l'ordinamento di tutte le righe TO DO;

3.2 Interferenze, Deadlock, Attesa attiva

TO DO

4 Costrutti di concorrenza

Per come è stata creata la struttura che immagazzina il puzzle e per come si è pensato di risolverlo, l'esecuzione in parallelo non necessita di particolare blocchi sull'oggetto in questione.
I costrutti che vengono utilizzate sono i seguenti.

4.1 Oggetto condiviso

TO DO

4.1.1 Synchronized

TO DO

4.2 wait()

TO DO

4.3 notifyAll()

TO DO