Mattia Sorgato, matricola 1004404

# Relazione di Progetto di Programmazione Concorrente e Distribuita

## Anno didattico 2014/2015

## Seconda consegna - Parte Concorrente

### Algoritmo di ordinamento Concorrente

L'ordinamento parallelo del puzzle nel mio progetto didattico è in qualche modo simile all'ordinamento sequenziale. Idealmente, nella sua realizzazione ho cercato di rendere parallelo l'ordinamento di ogni riga, cosicché il carico di ogni Thread fosse equo e ben distribuito.

Quindi, per ogni riga del puzzle, l'algoritmo crea un nuovo **SortLineThread** che si occupa di ordinare una singola riga, salvando il risultato dell'ordinamento su un oggetto condiviso con gli altri Thread, nella riga che gli compete.

Proprio per questa suddivisione in righe, la scrittura del risultato nell'oggetto condiviso non causa *race condition*, in quanto ogni Thread è responsabile di una singola riga e scrive solamente su quella.

Per ottenere questo risultato, però, è necessaria una prima passata iniziale sequenziale, per ricavare il primo membro di ogni riga e quindi il numero stesso di **SortLineThread** da creare.

Come si può intuire, nel processo di ordinamento, oltre al Main Thread sono attivi un numero **N** di Thread, con **N** il numero di righe del puzzle da ordinare.

### Classe Interna SortLineThread

Questa classe, come si può intendere dal nome, estende la classe Thread ed esegue l'ordinamento di una singola riga.

Si può notare dal codice come ogni campo dato della classe interna sia marcato **final**, soprattutto per il campo **private final List<Piece> puzzle**, che contiene il riferimento al puzzle disordinato ricevuto in input dall'utente. Questo garantisce quindi che avvengano solamente letture da questa lista, sebbene concorrenti, eliminando alla radice un possibile problema di *race condition* sul puzzle in entrata.