**Relazione PCD1718 Assigment #1 “The Game of Life”**

**Analisi del problema**

L’obiettivo dell’Assigment è l’implementazione in Java di una versione concorrente del gioco “The Game of Life” usando la programmazione concorrente multi-threaded.

Il gioco consiste nel calcolare e visualizzare l’evoluzione della matrice di celle che caratterizza il gioco, come sequenza di fotogrammi (ognuno dei quali rappresenta lo stato del mondo).

Nella matrice, ogni cella può essere in uno di due stati possibili, *live* e *dead*.

Dato lo stato s(t) della matrice, lo stato s(t+1) si computa con le seguenti regole:

* una cella m[i,j] che nello stato s(t) è *live* e ha zero o al più una cella vicina *live* (e le altre *dead)*, nello stato s(t+1) diventa *dead*(“muore di solitudine”)
* una cella m[i,j] che nello stato s(t) è *live* e ha quattro o più celle vicine *live*, nello stato s(t+1) diventa *dead*(“muore di sovrappopolamento”)
* una cella m[i,j] che nello stato s(t) è *live* e ha due o tre celle vicine *live*, nello stato s(t+1) rimane *live*(“sopravvive”)
* una cella m[i,j] che nello stato s(t) è *dead* e ha tre celle vicine *live*, nello stato s(t+1) diventa *live*

Il gioco deve presentare un’interfaccia grafica con pulsanti “start” e “stop” con cui si fa partire e fermare il gioco. Ogni stato del gioco deve essere visualizzato, insieme al numero di celle nello stato “live”.

**Indicazioni - requisiti**

* Il programma deve funzionare anche con matrici di dimensioni significative - es: (5000x5000)
* Massimizzare lo throughput, minimizzando il tempo di calcolo di ciascun fotogramma ed eventualmente anche della sequenza di fotogrammi
* Massimizzare la reattività della GUI
* Studiare e implementare meccanismi di coordinazione/sincronizzazione basati su semafori o monitor

**Descrizione della soluzione proposta**

**Architettura del sistema**

L’architettura del sistema è stata scomposta in tre livelli utilizzando il pattern Model View Presenter usando il modello ad Interattori, di conseguenza vi sono tre moduli: il dominio applicativo, gli interattori e la vista.

- Nel primo livello vi è una rappresentazione object oriented delle componenti del problema.

- Nel secondo livello è stata incapsulata la logica di aggiornamento del gioco.

- Nel terzo livello viene organizzata l’interfaccia grafica che permette la visualizzazione del gioco e l’interazione con l’utente.

**Implementazione**

**1 – Dominio applicativo**

Abbiamo implementato il dominio applicativo attraverso due classi: Cell e Board.

L’enum Cell rappresenta lo stato di una cella della scacchiera che può essere Alive o Dead.

La classe Board rappresenta la scacchiera di gioco. Questa è caratterizzata da un’altezza e una larghezza variabili e da la possibilità di recuperare/impostare lo stato di una cella date le coordinate x e y.

**2 – Interattori**

Abbiamo implementato un interattore “**BoardUpdater**” che si occupa dell’aggiornamento della scacchiera di gioco. Di questo interattore ne esistono due versioni, una sequenziale e una concorrente.

La versione sequenziale “**SequentialBoardUpdater**” viene eseguita sul thread corrente senza la creazione di nuovi thread.

Nella versione concorrente “**ConcurrentBoardUpdater**” è possibile configurare il numero di Worker (ognuno dei quali verrà eseguito su un thread indipendente).

Il metodo start si occupa della creazione di un nuovo thread per ogni Worker e di eseguirlo.

Il metodo stop si occupa di terminare l’esecuzione dei Worker.

Il metodo update gestisce l’aggiornamento della scacchiera di gioco (board). Per prima cosa si occupa di creare una nuova scacchiera che rappresenterà lo stato della scacchiera corrente dopo l’aggiornamento. Successivamente divide il lavoro fra i Worker cercando di bilanciarlo il più possibile. Dopodiché sospende la propria esecuzione in attesa che tutti i Worker abbiano terminato l’aggiornamento della porzione di scacchiera assegnatagli.

Ogni **Worker**, appena messo in esecuzione, aspetta che gli venga assegnata la porzione di scacchiera da aggiornare. Una volta terminata l’attesa, se ancora in esecuzione, effettua l’aggiornamento delle celle seguendo le regole del gioco (indicate nell’analisi del problema). Una volta terminato il lavoro, il Worker lo segnala al CuncurrentBoardUpdater e si mette in attesa di essere eseguito nuovamente.

Per ogni Worker la sincronizzazione con CuncurrentBoardUpdater è effettuata tramite due semafori binari: il primo *startUpdate* viene utilizzato per segnalare al Worker che può iniziare l’aggiornamento della propria porzione di scacchiera; il secondo *finishedUpdate* viene utilizzato per segnalare al CuncurrentBoardUpdater che il Worker ha terminato l’aggiornamento della sua porzione di scacchiera.

In questo modo l’updater segnala ad ogni Worker che la relativa porzione di scacchiera è pronta per essere aggiornata. Così attende che ogni Worker finisca il proprio lavoro per poi restituire la nuova scacchiera.

**3 – Interfaccia utente**

L’interfaccia utente è stata realizzata con Java FX ed è costituita da due schermate: nella prima si permette all’utente di settare le dimensioni della scacchiera di gioco (altezza e larghezza) ed il numero di Worker. Nella seconda si visualizza la scacchiera e sono presenti due pulsanti “start” e “stop”. Lo *start* permette l’avvio e la pausa del gioco, mentre lo *stop* la terminazione.

Utilizzando il pattern Factory sono stati incapsulati in una apposita classe le funzioni per la costruzione delle finestre grafiche. Le callback di queste finestre sono definite da opportuni Presenter che incapsulano la logica di controllo dei componenti (form, pulsanti, …) di cui sono costituite.

La computazione per l’update della scacchiera è effettuata su un thread separato “**GuiUpdater**” (task di Java FX) e questo permette di mantenere l’interfaccia grafica (gestita dal Java FX Application Thread) reattiva e in grado di intercettare l’input dell’utente.

La scacchiera di gioco è stata realizzata utilizzando un Canvas sul quale vengono disegnati i pixel attraverso la classe PixelWriter di Java FX: una classe ottimizzata per il disegno dei singoli pixel.

**Dinamica del sistema - Reti di Petri**

**Performance**

**Java Path Finder**