ConwayLifeActorsQak

Qui si descrive come realizzare il gioco della vita di Conway rappresentando anche le celle come attori Qak.

Questa può sembrare una inutile complicazione, ma vi alcuni validi motivi per proedere in questo senso:

1. l':analisi e la progettazione dei sistemi distribuiti (si veda *Sistemi distribuiti e microservizi (Koltraka Kevin)*) è (molto) **più complicata** di quella dei sistemi centralizzati. Pensare alle celle come attori introduce nuove problematiche, che ricorrrono quasi sempre anche nel caso di sistemi distribuiti e a microservizi.
2. l'uso di **modelli eseguibili del sistema** (Si veda: *Il metamodello Qak*), prima in locale e poi nel distribuito, permette di ridurre i tempi della analisi e della progettazione e di testare le soluzioni proposte.
3. In particolare, può essere difficile completare al primo colpo l'analisi di un problema complicato La possibilità di realizzare prototipi in tempi rapidi consente di capire meglio i requsiiti e il problema stesso, aprendo la via a uno sviluppo evolutivo e incrementale che può riguardare anche i requisiti e l'analisi del problema.
4. L'esperienza maturata nel realizzare *ConwayLife a celle distribuite* può aiutarci a qualche riflessione su come si possono organizzare **spettacoli con droni sincronizzati** che formano figure (dinamiche) nel cielo notturno Si veda: *Non droni ma Led*

Analisi del problema LifeActorsQak

Cominciamo con l'introduzione di alcune problematiche che si evidenziano subito.

1. Vicinanza tra celle: informazione logica o relatova alla dislocazione fisica?
2. Numero delle celle: chi lo stabilisce?
3. Scomparsa della struttura-dati grid: corrisponde al superamento della 'architettura monolitica' nel mondo a microservizi e implica comunicazioni via rete:
   - lo stato delle ccelle limitrofe a una caella deve essere reso noto mediante messaggi
   - il nuovo stato di una cella deve essere reso noto alle altre mediante messaggi; ma quando?
4. Denominazione delle celle: come fa ciascuna cella-actor ad avere un nome univoco nel sistema?. Chi stabilisce questo nome? E con quale criterio?
5. Evoluzione delle epoch: chi decide che una epoch è terminata e si può passare al calcolo della successiva?
6. Controllo del sistema: chi reagisce a comandi quali START/STOP/CLEAR della versione precedente?

Iniziamo dalla cella

Cominciamo l'analisi focalizzando l'attenzione sulle celle.

1. I componenti. Il sistema è composto da **NC=NRxNC** celle-actor.
2. I nomi. Ogni cella-actor deve avere nome univoco. La struttura del nome può essere definita in modo che ogni cella possa sapere, dato il suo nome, il nome delle celle limitrofe. Esempio di nome: cell_X_Y, con **0<=X<NR**, **0<=Y<NY**.
3. Le dimensioni. Pur essendo **NC** limitato, un valore di **NR** e **NC** troppo basso potrebbe rendere il gioco non significativo. Si possono proporre i segeunti valori:

   - **NR=NC=3** per un primo prototipo
   - **NR=NC=5** per un secondo prototipo
   - **NR=NC=20** per un prototipo simile al caso concentrato

   I valiri di **NR** e **NC** andrebbero fissati in un file di configurazione del sistema
4. Le azioni. Ogni cella-actor deve
   - rendere visibile (comunicare?) il suo stato corrente alle celle limitrofe
   - acquisire lo stato delle sue celle limitrofe
   - calcolare il suo nuovo stato nella prossima *Epoch*, in base alle regole Conway
   - ripetere le azioni precedenti dopo che la *Epoch* corrente si è stabilizzata

Rendere visibile lo stato di una cella

Ci sono due possibilità:

1. Comunazioni dirette: la cella **C** comunica direttemente con le celle limitrofe. In questo caso occorre individuare il giusto tipo di messaggi e di interezione.
2. Risorse osservabili: la cella è una risorsa che rende osservabile il suo stato alle celle interessate. In questo caso, ci sono due pattern principale.
   - Osservare Eventi: la cella **C** emette un evento che ha come payload il suo stato
   - Pattern Observer distribuito: le celle limitrofe si 'registrano' come observer alla cella **C**.

Acqusire lo stato di una cella

Questa problematica è legata alla precedente *Rendere visibile lo stato di una cella*.

Evoluzione delle Epoch

La evoluzione del gioco (in quanto risultato del comportamento di un sistema distribuito) può essere ottenuto in due modi principali:

- Coreografia: le celle **si coordinano** in modo da sapere quando una Epoch è termiata e quindi quando poter inziare le azioni relative alla Epoch successiva
- Orchestratore: si introduce un **orchestratore**, una sorta di direttore di orchestra che ha la responsabilità di 'dettare i tempi', cioè di capire quando una Epoch è termiata e quando indicare alle celle che è possibile inziare le azioni per una Epoch successiva

Controllare il sistema

Controllo del gioco: Le celle non devono solo eseguire il gioco, ma devono essere anche sensibili a forme di controllo quali:

- modifica del valore corrente di stato (fase di **inizializzazione**, **CLEAR**)
- attivazione delle azioni (**START**)
- sospensione delle azioni (**STOP**)
- terminazione delle attività (**EXIT**)

Queste forme di controollo nascono al di fuori del gioco e richedono che le celle siano sensibili a comandi provenienti dal mondo esterno ed emessi da un componente che possiamo denominare **gamecontroller**.

Costruzione del sistema

Le **NC=NRxNC** celle-actor possono essere create in due modi diversi:

- NC nodi reali: ogni cella 'nasce' su un suo nodo di elaborazione fisico distinto, ad esempio su un PC o su un RaspberryPi. In questo seocndo caso un **LED** potrebbe indicare il valore corrente dello stato della cella
- NC nodi simuati: ogni cella 'nasce' all'interno di uno stesso nodo fisico di elaborazione come **risorsa puramente logica**. In questo caso la creazione del sistema può essere realizzara da un opportuno programma, con notevole risparmio di risorse.

Quallo dei nodi simulati è l'approccio sicuramente più appropriato in fase di prima prototipazione. Il codice delle celle-actor messo a punto in questa fase può poi essere riusato per impostare il caso dei nodi reali, con il vantaggio di essere già stato sperimentato e testato da un punto di vista logico.

Risultato della analisi

Come analisti del problema, osserviamo che

- occore pervenire in tempi rapidei alla realizzazione di un primo prototipo che permetta di interagire con il committente per l'assestamento dei requisiti.
- il sofware del primo prototipo dovrebbe essere organizzato in modo da poter essere la base per successive evoluzioni e perfezionamenti
- introdurre un orchestratore per gestire la evoluzione del gioco riduce la complessità del sistema rispetto all'approcio coreografato e introduce un componente che può essere usato anche come controller del gioco
- la scelta delle forme di comunicazione/interazione tra le celle richiede un approfondimento della analisi

Come analisiti, proponiamo di impostare un quindi di impostare un primo prototipo a nodi simulati con i seguenti componenti (actor)

1. un **gamebuilder** che abbia il compito di creare i nodi simulati
2. un **gamecmaster** che funga da orchestratore del gioco
3. un **gamecontroller** (eventualemente coincidente con il **gamecmaster**) che abbia la responsabilità di realizzare il controllo sul gioco

Inoltre, sii ritiene opportuno:

1. definire inzialmente **NR=NC=3** e, dopo la fase di testing del primo prototipo, mostrare come si possa agenvolmente passare a valori più elevati del numero delle celle
2. preparare un RaspberryPi - con un LED - che possa diventare il supporto di elaborazione per una cella che opera su un proprio nodo fisico
3. capire come il prototipo realizzato possa evolvere verso un sistema di celle su nodi fisici.