Assignment 01
Concurrent Agent-based Simulations

Samuele De Tuglie
samuele.detuglie@studio.unibo.it

Emanuele Artegiani
emanuele.artegiani@studio.unibo.it

Pablo Sebastian Vargas Grateron
pablo.vargasgrateron@studio.unibo.it

April 7, 2024

1 Analisi del problema

Nel codice base fornito per il compito, è presente un simulatore che si occupa di modellare il traffico automobilistico, con e senza semafori. Il funzionamento del simulatore è il seguente: viene avviato un ambiente che genera le automobili e gestisce il loro movimento nel tempo. Ogni passo della simulazione rappresenta un intervallo temporale durante il quale l'ambiente elabora il comportamento di ciascun veicolo.

Il problema nel codice base risiede nel fatto che tutte le automobili simulate sono gestite da un unico thread, il quale si occupa anche dell'ambiente circostante. Questa configurazione potrebbe causare una simulazione lenta e inefficiente, specialmente in presenza di un elevato numero di entità in esecuzione.

2 Strategia risolutiva e architettura proposta

Per migliorare l'efficienza delle simulazioni si vuole modificare l'architettura del programma da sequenziale a concorrente, facendo in modo da sfruttare al massimo l'hardware disponibile parallelizzando il maggior numero di operazioni possibili.

Nella nuova implementazione del simulatore, si intende modellare la gestione dell'ambiente mediante l'utilizzo di un thread dedicato, con ciascuna automobile gestita da un thread che si occupa solo di quest'ultima (esempio: in una simulazione con N auto si avranno in totale $N + 1$ threads, ovvero N threads che si occupano appunto delle auto e uno che si occupa dell'ambiente).

A questo punto l'unica problematica è il coordinamento dei threads, che va risolta in modo tale da mantenere la medesima consistenza che si ha con il programma sequenziale.

2.1 Monitor

Per risolvere la problematica del coordinamento, si propone l'utilizzo di un monitor per gestire il corretto movimento di tutte le automobili e l'avanzamento del tempo nel simulatore. L'approccio consiste nell'implementare un unico monitor (Fig. 1) che supervisiona e garantisce che, ad ogni passo della simulazione, tutte le auto presenti abbiano completato il proprio movimento.

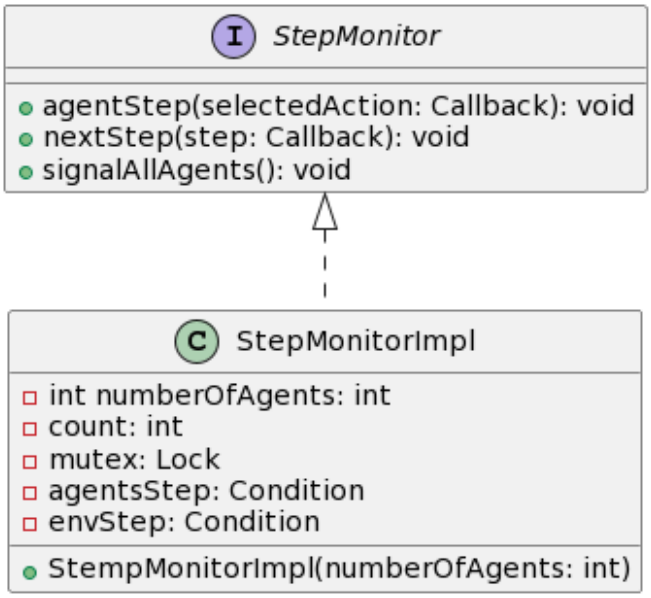

Figure 1: UML dell'interfaccia StepMonitor.java e la sua implementazione.

2.2 Simulazione, automobili e ambiente

Per garantire il coordinamento dei threads e l'atomicità delle operazioni è stato necessario creare delle nuove classi, modificate in modo tale da sfruttare il monitor proposto per raggiungere il risultato corretto in modo più efficiente:

- ***simtrafficconc.ConcurrentCarAgent***: estende la classe *CarAgent* e le uniche differenze da quest'ultima sono la presenza di un campo di tipo StepMonitor e l'implementazione del metodo *step* in maniera tale da usare il monitor.
- ***simtrafficconc.CarAgentThread***: rappresenta il thread che si occupa dell'esecuzione delle operazioni di *ConcurrentCarAgent*. Ogni *CarAgentThread* contiene un'istanza di *ConcurrentCarAgent*.
- ***simtrafficconc.ConcurrentRoadsEnv***: in maniera analoga a *simtrafficconc.ConcurrentCarAgent*, ha un campo StepMonitor e lo usa nell'implementazione di *step*.
- ***simengineconc.ConcurrentAbstractSimulation***: come la versione sequenziale (che estende) si occupa di inizializzare l'ambiente e gli agenti, ma in seguito a questo avvia i threads degli agenti e non prosegue eseguendo la simulazione direttamente ma la delega ad un altro thread (*SimulationThread*).
- ***simtrafficconc.SimulationThread***: si occupa dell'esecuzione della simulazione in modo concorrente, con l'utilizzo di StepMonitor.

2.3 Randomness

Nelle simulazioni di traffico introdotte è stato inserito un elemento di casualità per quanto riguarda la generazione dei valori dei parametri di movimento delle automobili. La classe che si occupa di generare i valori casuali permette la riproducibilità, in quanto utilizza un seme per la generazione di questi ultimi.

Tutto ciò permette di effettuare simulazioni più variegate e con risultati differenti, mantenendo lo stesso ambiente. Inoltre, è possibile ottenere simulazioni realistiche, ovvero con risultati non esclusivamente "positivi". Ad esempio, data una simulazione con la presenza di semafori su una o più strade, alcune automobili potrebbero passare col rosso in quanto il valore del loro parametro di decelerazione è troppo basso per fermarsi in tempo.

2.4 GUI

Una parte significativa del sistema è costituita dall'interfaccia grafica utente (GUI), che consente agli utenti di interagire con il programma in modo intuitivo e semplice. La GUI è stata progettata per fornire un'esperienza utente fluida e comprensibile.

Una volta avviata la GUI, l'utente si trova di fronte a due pulsanti principali: "Start" e "Stop", che consentono di avviare e interrompere la simulazione, rispettivamente. Inoltre, è presente un campo di testo in cui è possibile specificare il numero degli step della simulazione e un altro campo in cui è possibile inserire un seed randomico per la riproducibilità dei risultati 2.

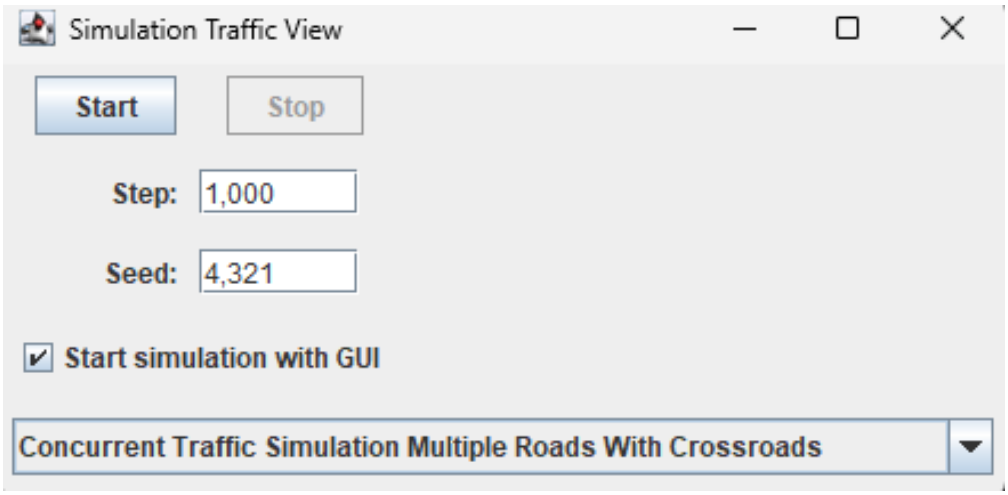

Figure 2: Interfaccia grafica all'avvio del programma.

Vi è anche una casella di controllo che permette all'utente di scegliere se abilitare o disabilitare l'utilizzo della grafica durante la simulazione. Questo è particolarmente utile per eseguire simulazioni più leggere o per risparmiare risorse di sistema. Infine, c'è un menu a tendina che permette all'utente di selezionare il tipo di simulazione da es-

eguire, fornendo una varietà di opzioni per adattare il programma alle specifiche esigenze dell'utente. Nel caso la simulazione viene avviata con la GUI allora verrà presentata una schermata simile a quella rappresentata in 3, altrimenti nel caso in cui non venga usata al termine della simulazione sarà presente una finestra che presenta i risultati ottenuti 4.

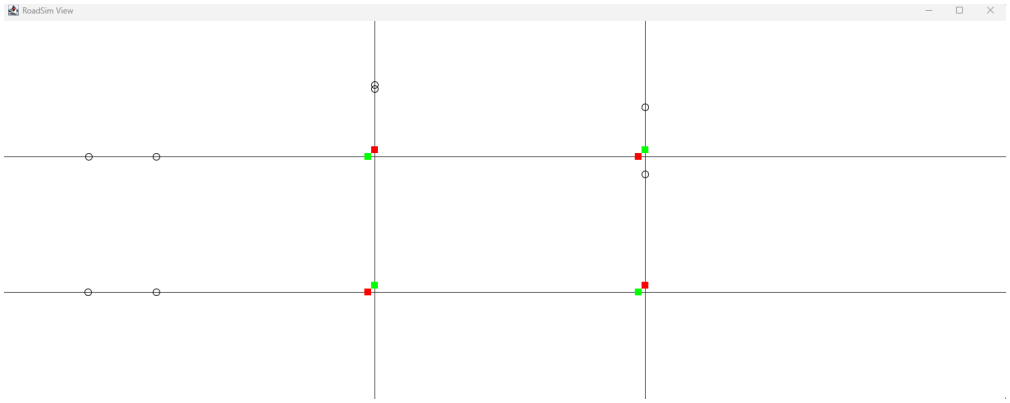

Figure 3: Esempio di simulazione con grafica.

Figure 4: Output della simulazione senza interfaccia grafica.

3 Comportamento del sistema

3.1 Descrizione

Il sistema che modella la simulazione si compone di due tipologie di agenti: *Environment* e *Car*. Gli agenti *Car* possono essere N.

In breve, il comportamento del sistema (Figura 5) è il seguente:

1. L'*Environment* esegue le proprie operazioni relative allo step corrente e si mette in attesa che gli agenti *Car* abbiano fatto lo stesso.

2. Una volta che l'*Environment* ha eseguito il proprio step, gli agenti *Car* eseguono concorrentemente le proprie azioni.

3. Una volta che tutti gli agenti *Car* hanno terminato, l'*Environment* procede ad eseguire le operazioni di fine step.

Tutto ciò viene ripetuto per il numero di step prefissato.

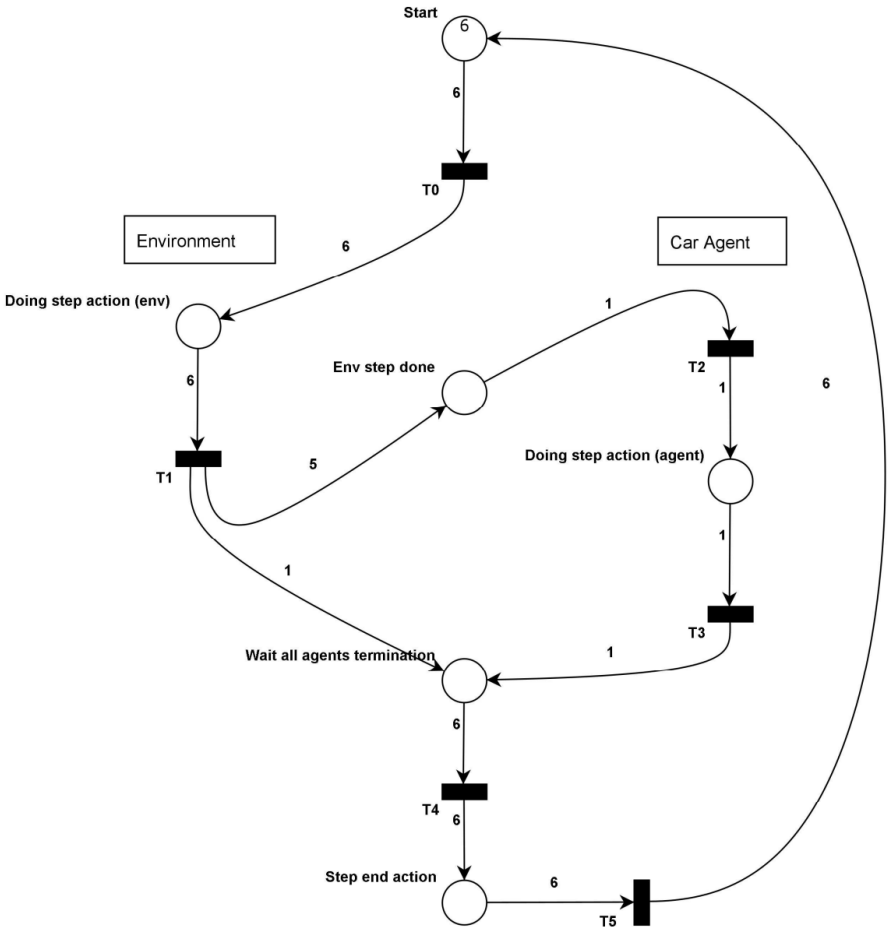

Figure 5: Rete di Petri che mostra il comportamento del sistema ad ogni step. In questo caso specifico sono presenti 5 *Car Agents*, ma il comportamento con N *Car Agents* è il medesimo. Il numero totale di token è N + 1.

4 Conclusioni

I test sono stati condotti su:

- CPU-1: Intel i7-10700k con frequenza di 5.2 GHz, dotata di 8 core e 16 threads.
- CPU-2: Intel i7-7700k con frequenza di 4.5 GHz, dotata di 4 core e 8 threads.

Sono state eseguite prove di correttezza utilizzando *Java Pathfinder* le cui relative configurazioni sono presenti nella cartella *src.main.jpf*. Tutte le prove terminano in modo

Simulazione	CPU	N.Steps	N.Cars	Seq	Conc
SingleRoadSeveralCars	CPU-1	1000	30	46810	482
WithCrossRoads	CPU-1	10000	4	347	354
SingleRoadMassiveNumberOfCars	CPU-1	100	5000	26528	4849
SingleRoadMassiveNumberOfCars	CPU-2	100	5000	28606	8534
MultipleCrossroadsMassiveNumberOfCars	CPU-1	100	5000	43613	7955
MultipleCrossroadsMassiveNumberOfCars	CPU-2	100	5000	49035	13193

Table 1: Tempi di esecuzione delle varie simulazioni di traffico in millisecondi (ms) in versione sequenziale (Seq) e concorrente (Conc).

corretto, non mostrando errori di alcun tipo come risultato.

Sono stati creati dei metodi per controllare che i risultati delle simulazioni sequenziali e concorrenti, eseguite con gli stessi parametri, dessero il medesimo risultato. Per fare ciò nel package *utils* sono presenti una classe che permette di salvare lo stato finale dell'ambiente e delle auto su file (o mostrarlo nella console) ed una che si occupa di eseguire un confronto dei suddetti file.

Come indicato nella Tabella 1, i tempi di esecuzione dei programmi concorrenti sono notevolmente inferiori rispetto a quelli dei programmi sequenziali, maggiore è il numero di entità eseguite nel simulatore. Si osserva che l'approccio con il monitor implementato offre vantaggi tanto più il numero di auto è elevato.
Questi risultati evidenziano che l'approccio concorrente in generale agevola l'esecuzione del programma, anche tenendo conto di caratteristiche come l'implementazione dei semafori.
Dalla Tabella 1 si può anche notare come il numero di core della CPU influenzi molto di più il tempo di esecuzione rispetto alla frequenza quando viene eseguito un programma progettato per essere concorrente.