PCD Assignment #01 - Concurrent Agent-based Simulations

v 0.9.1-20240318

Si vuole realizzare un programma concorrente che funga da framework di base per eseguire *simulazioni agent-based*, a partire da un'implementazione sequenziale fornita come esempio.

Descrizione del contesto e background

I modelli/simulazioni agent-based consistono nel modellare un certo fenomeno (o realtà o sistema complesso) strutturalmente come un insieme dinamico di entità dette *agenti,* che interagiscono all'interno di un certo *ambiente*[1]. Un agente rappresenta un'entità dotata di comportamento *autonomo*, ovvero incapsula la logica delle decisioni in merito a quali *azioni* compiere nell'ambiente (come output), a valle delle *percezioni* ottenute dall'ambiente stesso (input). Le percezioni e azioni di un agente dipendono da *dove è situato* l'agente stesso nell'ambiente, secondo un modello topologico che dipende dalla realtà modellata. La simulazione consiste nel determinare, a partire da un tempo logico iniziale `t0` (ove il modello di tempo dipende dalla specifica simulazione), come evolve il sistema a passi discreti di tempo `dt`, determinando ad ogni step lo stato aggiornato dell'ambiente e degli agenti. Come per le simulazioni in generale, questo è utile per studiare la dinamica di sistemi complessi, fare previsioni circa il loro comportamento in situazioni specifiche, etc.

Un programma in pseudocodice, sequenziale, che cattura il comportamento di una tipica simulazione ad agenti è il seguente:

```
/* data structures about agents and the environment */

agents: list of agents
env: environment


/* simulation initialisation */

t := t0
numSteps := 0;

env.init()
for each agent ag in agents
  ag.init()
```

[1] Agent-based modelling and simulation è un argomento di ricerca vasto, con ampia letteratura e sperimentazioni relativamente allo studio di sistemi complessi in domini diversi. Fra le piattaforme più usate: NetLogo. Qui un esempio (fra i tanti) relativo alla modellazione del traffico.

```
/* simulation execution of a number of steps */

while (numSteps < NumMaxSteps) {

  /* at each step, compute the state of the environment and the agents */

  env.step(dt)
  for each agent ag in agents
    ag.step(dt)

  t := t + dt
  numSteps++

  /* update possible statistics and views */

  updateStatistics(agents, env)
  updateViews(agents, env)
}
```

Il metodo `step` definisce il comportamento degli agenti (e dell'ambiente) ad ogni istante discreto di tempo. Per un agente, tale comportamento, in generale, è caratterizzato da tre fasi:

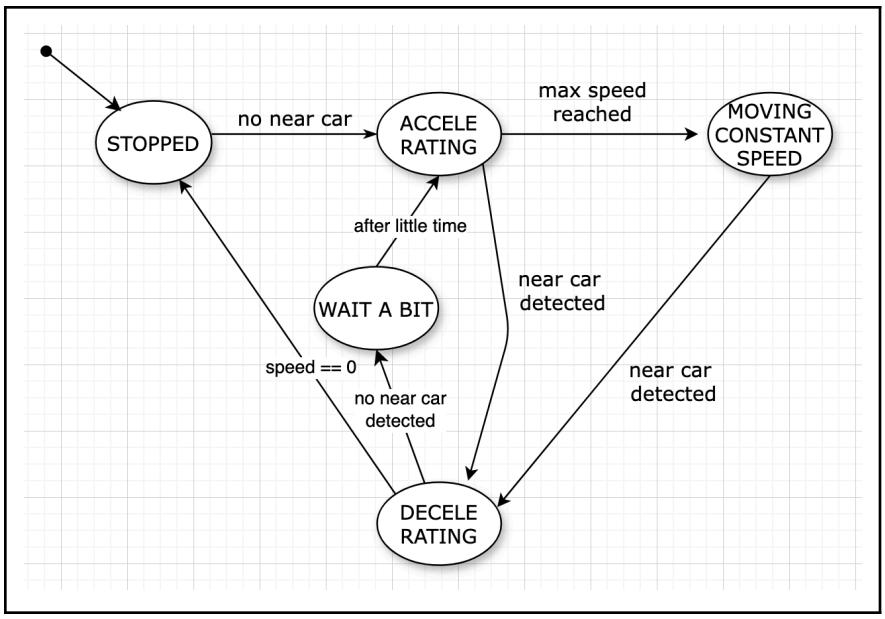
- una fase in cui l'agente determina le percezioni correnti (fase *sense*)
- una fase in cui, date lo stato corrente e le percezioni correnti, determina quale sia la prossima azione da fare (fase *decide*)
- una fase in cui in cui l'azione è eseguita sull'ambiente (fase *act*)

Nel materiale è fornito un semplice esempio di framework di base, sequenziale, per eseguire simulazioni agent-based (package `pcd.ass01.simengineseq`). Include un insieme di classi utili per modellare ed eseguire simulazioni concrete: `AbstractAgent, AbstractEnvironment e AbstractSimulation.`

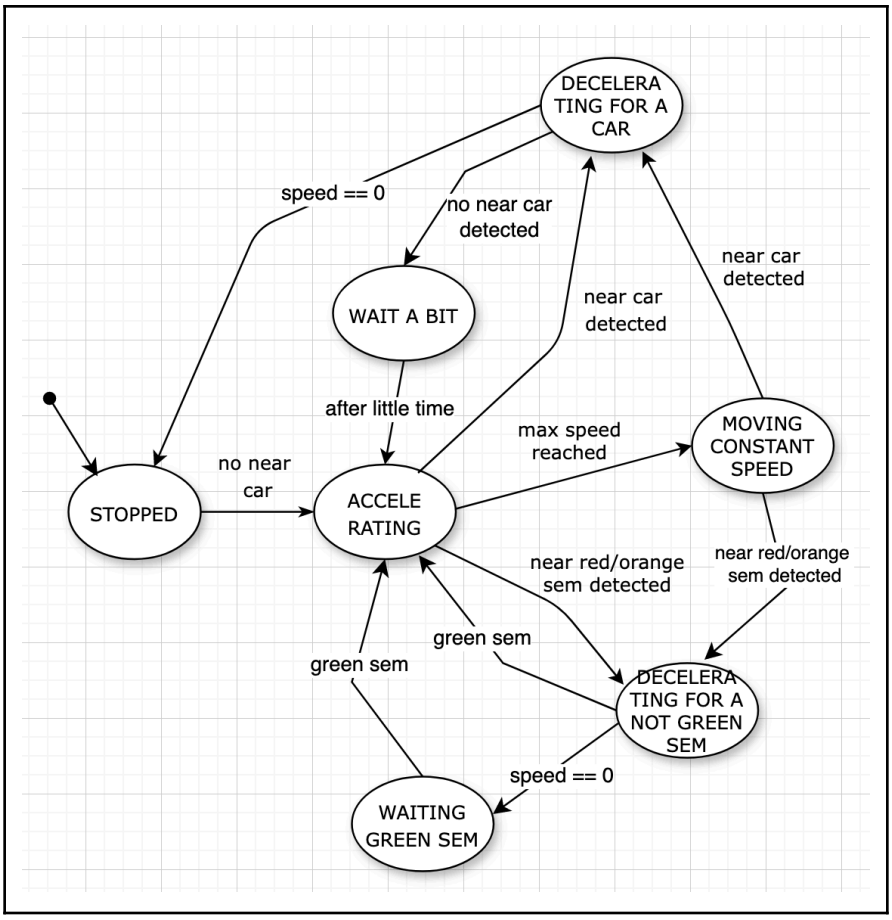
- `AbstractSimulation` è la classe di riferimento per l'assignment, contenente l'implementazione dell'algoritmo sequenziale con cui viene eseguita la simulazione (metodo `run`).

Come simulazioni concrete, sono forniti esempi semplici relativi al traffico.

- Il package `pcd.ass01.simtrafficbase` contiene le classi relative all'ambiente e agli agenti
  - `RoadsEnv` modella un ambiente costituito da strade (classe `Road`) e semafori (`TrafficLight`). Strade a senso unico, a singola corsia.
  - `CarAgent` è la classe base astratta che modella il comportamento base di un'automobile, con due implementazioni concrete
    - `CarAgentBasic` – comportamento in cui un'auto semplicemente regola la propria velocità con cui percorre la strada in relazione alla presenza di altre auto davanti. Il comportamento è descritto dalla seguente macchina a stati finiti:



- `CarAgentExtended` – comportamento che estende il precedente considerando anche i semafori. La macchina a stati finiti è una estensione della precedente:



- il package `pcd.ass01.simtrafficexamples` contiene esempi concreti di simulazioni, che possono essere lanciate dalla classe `RunTrafficSimulation`
    - `TrafficSimulationSingleRoadTwoCars` - singola strada senza semafori, con due auto
    - `TrafficSimulationSingleRoadSeveralCars` - singola strada senza semafori, con diverse auto
    - `TrafficSimulationSingleRoadWithTrafficLightTwoCars` - singola strada con semafori, con due auto
    - `TrafficSimulationWithCrossRoads` - due strade con incrocio (due semafori)

Nel package `pcd.ass01.simtrafficexamples` ci sono due esempi di classi che implementano una semplice visualizzazione (`RoadSimView`) e una semplice forma di statistica (`RoadSimStatistics`), agganciati alla simulazione come listener.

- Nel caso si voglia attivare la visualizzazione, è fornita la possibilità di sincronizzare l'esecuzione della simulazione con il tempo reale, fissando il numero di frame al secondo
    - metodo `syncWithTime`, da usare nel setup delle simulazioni;

La consegna

La consegna si articola in due punti, più uno facoltativo:

1) Realizzare una versione concorrente del framework per simulazioni agent-based, al fine di sfruttare in modo opportuno l'hardware multicore per migliorare le performance, in particolare nel caso in cui si considerino simulazioni che coinvolgono un numero considerevole di agenti. Il nuovo framework può prendere spunto da quello sequenziale fornito, inclusi gli esempi, tuttavia può rivederne struttura e organizzazione nel modo che si ritiene più opportuno.

   Si chiede di eseguire prove di performance al fine di verificare la scalabilità della soluzione adottata.
   - Per fare prove di performance, nel package `pcd.ass01.simtrafficexamples` è fornita la simulazione `TrafficSimulationSingleRoadMassiveNumberOfCars` in cui si considerano 5000 auto e la classe `RunTrafficSimulationMassiveTest` che manda in esecuzione la simulazione per un certo numero di step (100), prendendo i tempi. Facoltativamente, si può considerare anche la realizzazione di nuove simulazioni più complesse, che – ad esempio – considerino molte strade con diversi incroci.

   Si richiede inoltre di eseguire verifiche di correttezza dell'implementazione (semplificata) o di un modello dell'implementazione mediante JPF.

   CONSIDERAZIONI UTILI ALL'ANALISI:

   - nelle fasi di *sense* e *decide* gli agenti non modificano l'ambiente, l'accesso è solamente in lettura
   - la modifica dell'ambiente avviene nella fase di act
   - la versione concorrente non deve alterare il comportamento della simulazione, che deve produrre i medesimi risultati (ma auspicabilmente in tempo inferiore). Questo implica che, ad esempio, l'ordine con cui vengono eseguite le azioni degli agenti sull'ambiente ad un certo istante di tempo dovrebbe essere totalmente indipendente dall'implementazione sequenziale o concorrente della simulazione

2) Estendere il programma al punto precedente includendo una GUI che permetta di gestire l'esecuzione delle simulazioni, in particolare includendo:

- pulsanti start/stop per avviare/fermare l'elaborazione
- Input box per specificare il numero di step da eseguire come parametro iniziale

3) *Facoltativo:* Negli esempi non sono stati considerati numeri casuali, che invece compaiono spesso nelle simulazioni. Formulare un esempio di simulazione in cui ci siano anche numeri casuali ed estendere il framework concorrente costruito al punto 1 al fine di gestire consistentemente anche simulazioni in cui si usino numeri casuali.

Il deliverable

Il deliverable consiste in una cartella “Assignment-01” compressa (formato zip) da sottoporre sul sito del corso, contenente:

- directory **src** con i sorgenti del programma
- directory **doc** che contenga una breve relazione in PDF (report.pdf). La relazione deve includere:
  - Analisi del problema, focalizzando in particolare gli aspetti relativi alla concorrenza.
  - Descrizione della strategia risolutiva e dell’architettura proposta, sempre focalizzando l’attenzione su aspetti relativi alla concorrenza.
  - Descrizione del comportamento del sistema (ad un certo livello di astrazione) utilizzando **Reti di Petri.**
  - In merito al primo punto:
    - Prove di performance e considerazioni relative
    - Verifiche di correttezza con JPF

NOTE IMPORTANTI

- Si richiede di adottare un approccio basato esclusivamente su programmazione multithreaded, adottando, da un lato, principi e metodi di progettazione utili per favorire modularità, incapsulamento e proprietà relative, dall’altro una soluzione che massimizzi le performance e reattività.
- Come meccanismi di coordinazione prediligere l'uso di monitor rispetto ad altre soluzioni di più basso livello.
- La soluzione deve esclusivamente considerare thread e monitor. Non è consentito l'uso di costrutti di più alto livello (es: framework Executors, Parallel Streams, Virtual Thread…), che

permettono di astrarre dalla gestione dei thread – questi verranno trattati nella prossima parte del corso.