Report Assignment 1

- ## Introduzione:

Il presente report fornisce un'analisi dettagliata degli aspetti relativi alla concorrenza nell'implementazione di un framework per simulazioni agent-based. L'obiettivo principale è garantire un'efficiente gestione della concorrenza tra gli agenti e l'ambiente condiviso all'interno del contesto della simulazione.

- ## Contesto e Problematica:

I modelli agent-based sono utilizzati per modellare sistemi complessi composti da entità autonome chiamate agenti che interagiscono all'interno di un ambiente condiviso. La simulazione di tali sistemi richiede la gestione efficace della concorrenza poiché gli agenti devono operare indipendentemente e contemporaneamente all'interno dell'ambiente, garantendo al contempo coerenza e integrità dei dati.

Per fornire un framework adeguato per le simulazioni agent-based è necessario soffermarsi su alcune considerazioni:

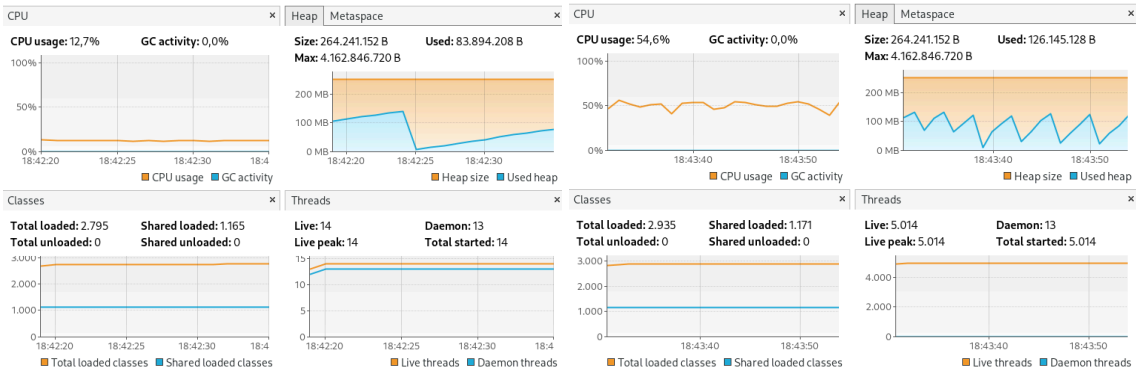
1. Ogni componente dovrebbe avere un'interfaccia chiara e definire il suo ruolo nel contesto della simulazione.
2. Gli agenti dovrebbero essere implementati come thread separati, ciascuno eseguendo la propria logica di simulazione in modo concorrente. E’ necessario assicurarsi che ogni agente interagisca con l'ambiente condiviso attraverso un'interfaccia sincronizzata.
3. L'ambiente condiviso dovrebbe essere protetto da accessi concorrenti utilizzando un monitor. Bisogna garantire che solo un agente alla volta possa accedere o modificare lo stato dell'ambiente.
4. Controllare come vengono coordinati gli agenti per evitare conflitti e inconsistenze nell'ambiente condiviso. Necessario tramite meccanismi di sincronizzazione all'interno del monitor segnalare quando è sicuro per un agente eseguire determinate azioni.

Di seguito, dopo una analisi delle prestazioni di entrambi i framework, è possibile confrontarli come segue:

- Il framework concorrente ha un utilizzo significativamente più elevato della CPU rispetto a quello sequenziale (54.6% rispetto a 12.7%). Ciò va a indicare

come il framework concorrente è in grado di sfruttare meglio le risorse della CPU, eseguendo più processi in parallelo e quindi ottenendo un maggiore utilizzazo della potenza di calcolo disponibile.

- Il framework concorrente ha un numero molto maggiore di thread vivi rispetto a quello sequenziale (5014 rispetto a 14). Questo suggerisce che il framework concorrente utilizza intensivamente i thread per eseguire operazioni parallele, consentendo una maggiore concorrenza e una più efficiente gestione delle risorse.
- Il framework concorrente ha un tempo di esecuzione molto più breve rispetto a quello sequenziale (11858 ms rispetto a 38187 ms). Questo indica che la versione concorrente è in grado di completare la simulazione molto più rapidamente, grazie alla sua capacità di eseguire più operazioni in parallelo.
- Il tempo medio per passo è significativamente inferiore nel framework concorrente rispetto a quello sequenziale (110 ms rispetto a 381 ms). Questo suggerisce che, nonostante l'aumento dell'utilizzo della CPU, il framework concorrente è in grado di completare ciascun passo della simulazione in modo molto più efficiente e rapido.



Framework sequenziale

Framework concorrente

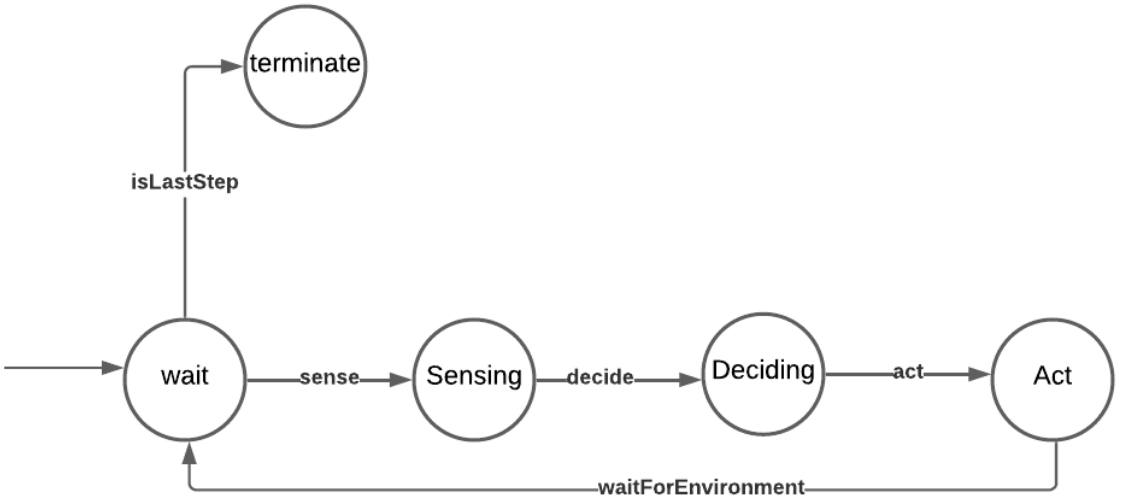
- Strategia risolutiva:

Le tre classi principali (AbstractAgent, AbstractEnvironment e AbstractSimulation) sono progettate per implementare una simulazione agent-based con un approccio concorrente, sfruttando thread e monitor per garantire la sincronizzazione e la coerenza dei dati tra gli agenti e l'ambiente condiviso.

La classe **AbstractAgent** rappresenta la base per la definizione dei tipi di agenti che partecipano alla simulazione. Ogni agente è un thread separato che esegue le fasi di percezione (sense), decisione (decide) e azione (act). Gli agenti vengono

sincronizzati sull'ambiente condiviso (AbstractEnvironment) durante la fase di azione per garantire la coerenza nell'accesso e nella modifica dello stato dell'ambiente.

- Ogni agente viene inizializzato con un identificatore univoco (myId) e l'ambiente (**AbstractEnvironment**) in cui opererà durante la simulazione.
- Il metodo run() definisce il comportamento dell'agente durante l'esecuzione come thread. All'interno del ciclo principale, l'agente esegue continuamente le fasi di percezione, decisione e azione finché non viene interrotto.
- Le fasi di percezione (sense) e decisione (decide) vengono eseguite sequenzialmente all'interno del ciclo principale, consentendo all'agente di ottenere informazioni sull'ambiente e di prendere decisioni basate su tali informazioni. Durante la fase di azione (act), l'agente interagisce con l'ambiente condiviso per eseguire un'azione. Questa fase è sincronizzata sull'ambiente mediante un blocco synchronized per garantire la coerenza nell'accesso e nella modifica dello stato dell'ambiente.
- Dopo ogni passo della simulazione, l'agente notifica all'ambiente che ha completato un passo (`env.agentStepped()`) e attende la sincronizzazione con l'ambiente prima di procedere al passo successivo (`env.waitForEnvironment()`).
  Se l'ambiente indica che è l'ultimo passo della simulazione (`env.isLastStep()`), l'agente interrompe il proprio thread.



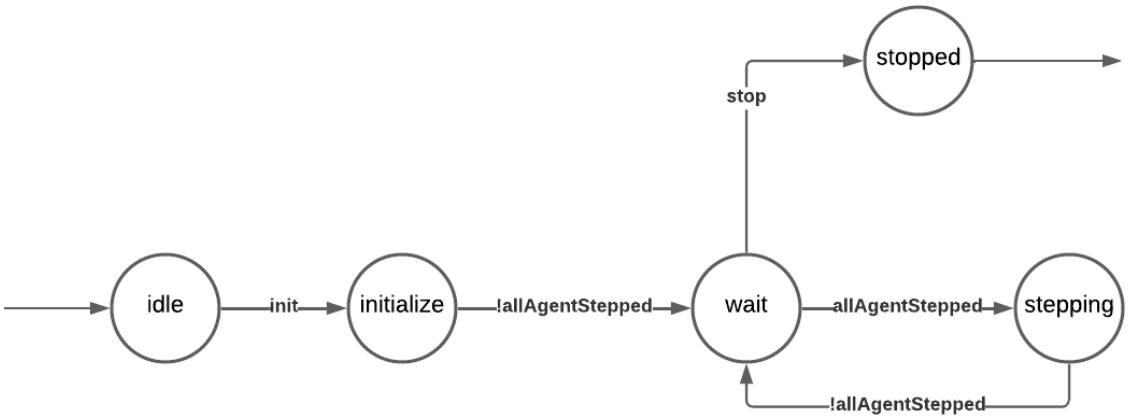
La classe **AbstractEnvironment** rappresenta la base per definire l'ambiente della simulazione agent-based. Questo ambiente fornisce le condizioni in cui gli agenti operano e interagiscono durante la simulazione.

Esecuzione dello step:

- Il metodo `executeStep(int dt, boolean isLastStep)` viene chiamato per eseguire un passo di simulazione con un certo intervallo di tempo (`dt`) e per verificare se è l'ultimo passo della simulazione. Questo metodo si occupa della sincronizzazione tra gli agenti durante l'esecuzione dei loro passi.
- Durante l'esecuzione di un passo di simulazione, l'ambiente controlla se tutti gli agenti hanno completato il loro passo precedente. Se non è così, l'ambiente attende utilizzando la condizione `allAgentsStepped`.
- Una volta che tutti gli agenti hanno completato il loro passo, l'ambiente esegue il proprio passo utilizzando il metodo astratto `step(int dt)`.
- Dopo aver eseguito il passo, l'ambiente segnala agli agenti che devono eseguire il prossimo passo e imposta il contatore degli agenti a zero.

Interazione con gli agenti:

- Gli agenti chiamano il metodo `agentStepped()` per segnalare all'ambiente che hanno completato il proprio passo.
- L'ambiente attende che tutti gli agenti abbiano completato il loro passo utilizzando la condizione `allAgentsStepped`, garantendo che tutti gli agenti abbiano un'opportunità di eseguire il proprio passo prima che l'ambiente proceda.
- Gli agenti possono richiedere i loro percezioni chiamando il metodo astratto `getCurrentPercepts(String agentId)`, e possono eseguire azioni sull'ambiente chiamando il metodo astratto `doAction(String agentId, Action act)`.



La classe **AbstractSimulation** fornisce un'implementazione di base per definire simulazioni concrete nell'ambito di un sistema agent-based. Ecco un'analisi della strategia risolutiva e dell'architettura proposta:

Ambiente e Agenti:

- L'ambiente della simulazione è rappresentato dall'istanza di AbstractEnvironment, mentre la lista degli agenti partecipanti è rappresentata dalla lista di oggetti AbstractAgent.

Configurazione della Simulazione:

- Il metodo setup() è astratto e viene utilizzato per configurare la simulazione specificando l'ambiente e gli agenti. Questo metodo deve essere implementato dalle sottoclassi per configurare l'ambiente e aggiungere gli agenti necessari alla simulazione.

Esecuzione della Simulazione:

- Il metodo run(int numSteps) esegue la simulazione per un numero specificato di passi. Utilizza un approccio sequenziale per eseguire ogni passo della simulazione.
- Durante ogni iterazione del ciclo while, viene eseguito un passo della simulazione chiamando il metodo executeStep() sull'ambiente e aumentando il contatore dei passi.
- Viene controllato se la simulazione è stata interrotta, in tal caso la simulazione viene arrestata prematuramente.

Gestione degli Agenti:

- Prima di iniziare la simulazione, ogni agente viene inizializzato chiamando il metodo init(AbstractEnvironment env) sull'ambiente.
- Durante l'esecuzione, ogni agente viene avviato utilizzando il metodo start(), che avvia un nuovo thread per ogni agente.
- Durante l'arresto della simulazione, vengono interrotti tutti gli agenti in esecuzione utilizzando il metodo interrupt().