ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea in Ingegneria e Scienze Informatiche

slime-mold Aggregation?

Tesi di laurea in:
Programmazione Ad Oggetti

Relatore
Prof. Mirko Viroli

Candidato
Lorenzo Tosi

Correlatore
Dott. Gianluca Aguzzi

IV Sessione di Laurea
Anno Accademico 2022–2023

Abstract

Max 2000 characters, strict. contesto + problema obiettivi e metodi contributi risultati conclusioni

Optional. Max a few lines.

Acknowledgements

Optional. Max 1 page.

Indice

Elenco delle figure

List of Listings

Capitolo 1

Introduzione

Nel vasto campo della ricerca scientifica, negli ultimi anni i comportamenti complessi emergenti sono oggetto di crescente interesse e studio. Questi fenomeni rappresentano il manifestarsi di comportamenti collettivi che sorgono dall'interazione dinamica di molteplici componenti di un sistema, difficilmente prevedibili se si considerano solamente le leggi che regolano il comportamento del singolo.

In natura questa caratteristica comportamentale è osservabile in un grandissimo numero di ambiti: si pensi, ad esempio, al regno animale, dove è possibile ritrovare speciali "forme" e comportamenti di stormi di uccelli oppure di banchi di pesci; lo stesso accade agli esseri umani in contesti come il traffico cittadino, il mercato della borsa valori o il gioco del poker.

Un esempio significativo di comportamento emergente è quello osservabile in biologia in una colonia di formiche. Nonostante le formiche, se considerate come esseri "singoli" seguano regole di comportamento molto semplici, l'interazione tra di esse dà origine ad una "comunità" omogenea e, seppure sia assente una struttura gerarchica, sono presenti una serie di modelli condivisi complessi per quanto riguarda la ricerca del cibo, la costruzione di nidi e la difesa del territorio. Ogni formica reagisce a degli stimoli, ovvero tracce chimiche provenienti da altre formiche e, al contempo, essa stessa lascia segnali agli altri membri della comunità: si crea così una reazione a catena che coinvolge tutte le formiche della colonia, che tendono a imitare il comportamento delle altre. Questo fenomeno è simile ad altre strutture emergenti presenti in natura e riscontrate sia negli "insetti sociali" (e.g. termiti,

vespe, api,...), ovvero insetti che formano colonie con mansioni diversificate, sia, in generale, in animali che vivono in gruppo (come pesci, tartarughe, mandrie di mammiferi,...). Questa tipologia di eventi, generalmente, si basa principalmente su feromoni o odori chimici.

Nel contesto scientifico, simulare in un ambiente protetto questo tipo di fenomeni è estremamente importante per diversi motivi:

- Comprenderne la complessità: i fenomeni complessi, come detto sopra, sono caratterizzati da interazioni dinamiche tra i componenti del sistema di riferimento. La simulazione diventa una risorsa chiave per esplorare, studiare e comprendere moltissimi aspetti di queste dinamiche e permette di osservare le interazioni dei diversi elementi in infiniti modi.

- Prevedere il comportamento del sistema: poiché questi fenomeni sono altamente aleatori, la simulazione può essere eseguita per cercare di prevedere e avere maggior consapevolezza del comportamento futuro di un sistema emergente in modo tale da poter prendere delle decisioni informate.

L'obiettivo di questa tesi è esplorare il fenomeno dell'aggregazione di questi organismi, sviluppando un sistema software che si interfacci ed utilizzi a pieno tutti gli elementi chiave del simulatore Alchemist. Quest'ultimo, infatti, permette di riprodurre eventi appartenenti a domini estremamente differenti tra loro, come simulazioni chimiche o il comportamento di pedoni in diverse situazioni.

La tesi presenta la seguente struttura:

- **Contesto**
- **Analisi**
- **Design**
- **Implementazione**
- **Evaluation**
- **Conclusioni**

Capitolo 2

Contesto

2.1 Alchemist

Alchemist [PMV13] è un simulatore DES (Discrete Event System) che estende il modello computazionale base delle reazioni chimiche in modo tale da favorirne l'applicabilità a situazioni complesse, pur mantenendo elevate prestazioni. In particolare, Alchemist si fonda su una versione ottimizzata dell'algoritmo di Gillespie[Gil77] chiamata Next Reaction Method[GB00], esteso in modo tale da poter lavorare con un ambiente agile e dinamico dove è possibile aggiungere o rimuovere reazioni, dati e connessioni topologiche. Le applicazioni già implementate sono varie e comprendono, ad esempio, simulazioni di reazioni biochimiche e movimento di pedoni. Il punto di forza del sistema è il meta-modello estremamente astratto, la cui effettiva realizzazione è demandata alle "incarnazioni", le quali rappresentano l'implementazione vera e propria delle diverse categorie di simulazioni eseguibili all'interno. Attualmente troviamo 4 incarnazioni:

- Protelis
- SAPERE
- Biochemistry
- Scafi

2.1.1 Il meta-modello

Come accennato in precedenza, il meta-modello è uno dei punti di forza maggiori di Alchemist. Per meta-modello si intende un tipo di paradigma che descrive la struttura, le regole e le relazioni che i modelli di dati devono seguire all'interno di un sistema. Rappresenta in modo astratto i concetti e le relazioni all'interno del dominio di interesse e stabilisce i vincoli e le convenzioni che tutti i modelli devono usare. Dunque, tutte le incarnazioni sviluppate presentano le stesse entità "base". Poichè Alchemist è sviluppato partendo da un'ispirazione orientata alla chimica/biochimica, le entità presentano nomi riconducibili a quei mondi. Infatti troviamo:

- **Molecole** (*molecule*): il nome di un dato, concettualmente interpretabile come il nome di una variabile.
- **Concentrazioni** (*concentration*): il valore associato alla molecola.
- **Nodi** (*node*): il "contenitore" di molecole e reazioni.
- **Ambiente** (*environment*): l'astrazione dello spazio; è un "contenitore" di nodi e svolge i seguenti compiti:
  - Restituire la posizione dei nodi.
  - Restituire la distanza tra due nodi.
  - Supportare il movimento dei nodi, se presente.
- **Vicinato** (*neighborhood*): un entità composta da un nodo centrale e un insieme di nodi vicini.
- **Regola di collegamento** (*linking rule*): una funzione relativa allo stato corrente dell'ambiente che associa ad ogni nodo un vicinato.
- **Reazione** (*reaction*): un qualsiasi evento che provoca un cambiamento dello stato dell'ambiente. È definita da una lista di condizioni, una o più lista di azioni e una distribuzione temporale. La frequenza con la quale avviene una reazione dipende da:

  - Un parametro statico “rate”.
  - Il valore di ogni condizione.
  - Una “rate equation”, ovvero una equazione che combina il parametro statico (rate) con i valori delle condizioni, restituendo un “instantaneous rate”.
  - Una distribuzione temporale.

- **Condizione** (*condition*): una funzione che, dato lo stato attuale del'ambiente (environment), restituisce un booleano ed un numero. Se il booleano è falso la reazione non può avvenire. In caso contrario, invece, avviene. Il numero può invece influenzare la velocità della reazione a seconda della reazione e della distribuzione temporale.

- **Azione** (*action*): un cambiamento nell'ambiente.

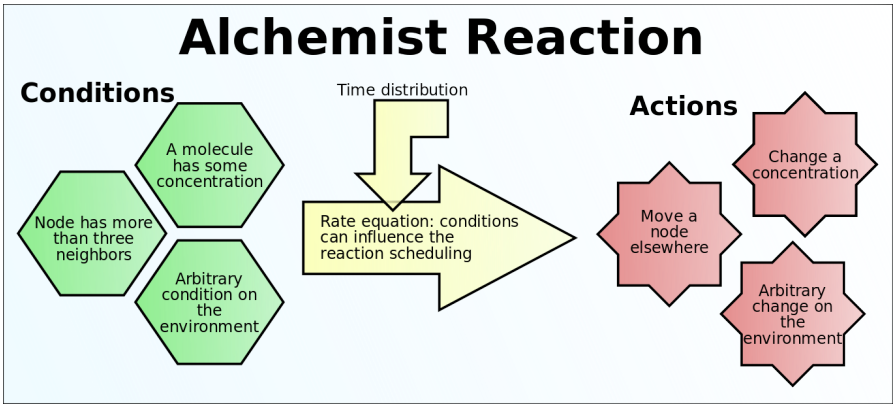

Figura 2.1: Rappresentazione di una reazione in Alchemist

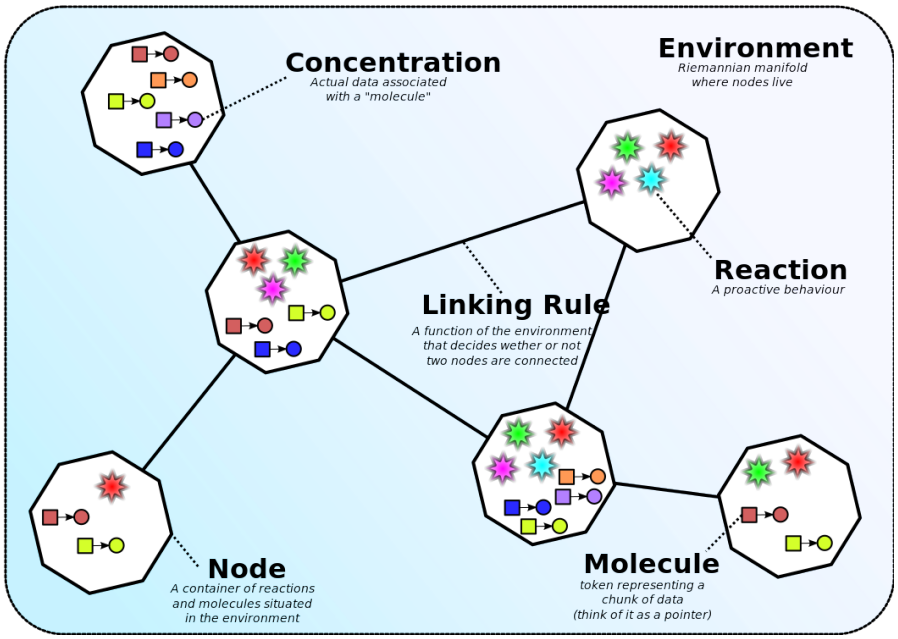

Figura 2.2: Il modello di Alchemist

2.2 Simulazione di riferimento

La simulazione utilizzata come riferimento per questo progetto di tesi è presente nella libreria di modelli di NetLogo [Wil97]. Il modello di riferimento è "Slime" [Wil97] e per simulare l'aggregazione di tanti singoli organismi in un gruppo si fa riferimento al comportamento delle tartarughe. Quest'ultime si muovono in uno spazio a griglia e durante il loro movimento rilasciano una particolare molecola chiamata "feromone" che si deposita in una posizione precisa. L'intero mondo è quindi suddiviso in tantissime "micro-aree" chiamate *patch*. La tartaruga per muoversi "annusa" davanti a se, ovvero percepisce se nelle *patch* vicine è presente del "feromone". Se il valore di quest'ultimo è abbastanza alto, la tartaruga si sposterà nella posizione "annusata", mentre, in caso contrario la tartaruga si muoverà in modo randomico nello spazio circostante. Durante tutto ciò, le *patches* diffonderanno del "feromone" alle varie posizioni vicine e, con il passare del tempo, il "feromone" tende ad evaporare (ovvero sparire) dalla griglia.

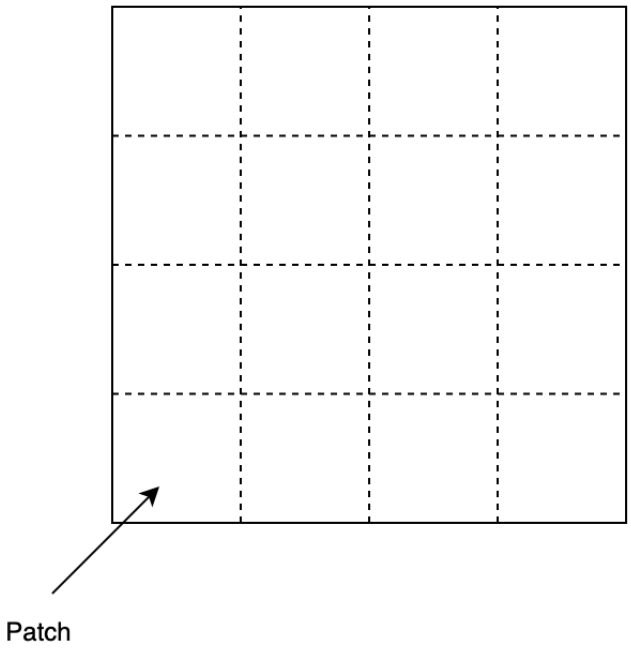

Figura 2.3: Il mondo suddiviso in patch

2.3 Overview su *slime-mold*

Nel contesto della presente tesi, sulla base della simulazione di riferimento2.2 e i comportamenti osservati in essa, si è scelto di indagare un altro fenomeno dalle caratteristiche estremamente simili: la muffa mucillaginosa. In natura, l'aggregazione delle cellule di muffa mucillaginosa (detti anche funghi mucillaginosi o, in inglese, *slime-mold*) rappresenta comportamento in cui entità individuali interagiscono tra di loro per formare strutture complesse e funzionali. Lo *slime-mold* è un organismo unicellulare ma, talvolta, può trovarsi anche ad agire come un'entità multicellulare. Pur non essendo un fungo è spesso classificato nella stessa categoria per via delle sue caratteristiche affini a quelle di questi organismi. La particolarità principale della muffa mucillaginosa è che può comportarsi come un organismo unicellulare o multicellulare a seconda delle diverse condizioni ambientali in cui si trova. L'habitat principale di questi organismi è il terreno umido, dove di solito si nutrono di foglie morte o di tronchi di alberi in putrefazione. Quando trova una fonte di cibo, lo *slime-mold* si aggrega, formando una massa citoplasmatica detta "plasmodio", composta da un grande numero cellule. Inoltre, questa massa può muoversi, "navigando" attraverso il terreno in cerca di cibo. Infatti, se le risorse alimentari scarseggiano o l'ambiente diventa meno ospitale, lo *slime-mold* può assumere forme diverse: può formare spore, resistenti per sopravvivere in condizioni avverse, oppure aggregarsi insieme ad altri individui simili per formare una struttura multicellulare che si comporta come un'unica entità, condividendone di conseguenza risorse e compiti.

Capitolo 3

Analisi dei requisiti

3.1 Requisiti

Analizzando il modello presente su NetLogo[Wil97], possiamo individuare le caratteristiche e i requisiti che la simualzione dovrà avere:

- Entità "vive", che si muovono e depositino il feromone.
- Un ambiente che gestisca la presenza del feromone; in particolare dovrà:
  - Permettere il depositarsi della sostanza.
  - Diffondere la sostanza.
  - Evaporare la sostanza.
- Le entità dovranno avere un concetto di direzione.
- Il movimento deve seguire delle regole ben precise.

3.2 Requisiti funzionali

A livello funzionale il progetto dovrà essere in grado di simulare l'aggregazione di singole entità all'interno di un ambiente. L'ambiente dovrà essere in grado di gestire la presenza di una sostanza, il feromone; in particolare dovrà essere in grado di gestire il depositarsi della sostanza in un punto, la diffusione della stessa in un

vicinato e l'evaporazione. Le entità dovranno essere "vive", ovvero dovranno essere in grado di muoversi liberamente. Inoltre dovranno essere in grado di rilasciare la sostanza nel punto in cui si trovano e di percepire la presenza della sostanza stessa.

3.3 Requisiti non funzionali

La logica di movimento non dovrà essere banale ma dovrà seguire delle regole precise in modo tale da poter simulare al meglio il comportamento di di un essere vivente.

Capitolo 4

Design

4.1 Layer

Si pensi alla simulazione come se fosse un micro-mondo, una "città" complessa e ricca di informazioni. È di interesse capire il livello di temperatura oppure di inquinamento nelle varie aree cittadine, dati invisibili all'occhio umano ma comunque presenti nell'ambiente e percepibili da chi lo abita. Si ha bisogno di inserire nell'ambiente degli "strati" invisibili che hanno il compito di raccogliere queste informazioni. È possibile, in Alchemist, definire questi "strati" di dati, chiamati Layer.
Nel contesto di questo progetto è stato necessario l'utilizzo di un Layer che avesse la funzione di "rete di raccolta" dei feromoni che, nella simulazione di riferimento 2.2, venivano rilasciati dalle tartarughe nelle varie posizioni dello spazio. In questo caso il layer ha la stessa dimensione dell'ambiente, in modo tale da poter suddividere l'intera area nelle varie *patch* di cui si è discusso sopra. Il layer, chiamato *PheromoneLayer* ha come compiti:

- Implementare una struttura dati per tenere traccia della quantità di feromone presente in ogni posizione.
- Offrire un modo per aggiornare i valori del feromone.
- Condividere con le altre classi la struttura dati contenente la quantità di feromone per una specifica posizione.

- Lasciare la possibilità all'utente di decidere le dimensioni dell'area totale di riferimento e di ogni *patch*.

4.1.1 Le posizioni

Un aspetto di particolare rilevanza è stato il processo decisionale relativo alla gestione delle posizioni collegate al deposito del feromone. Nella simulazione di riferimento 2.2 si trova uno spazio a griglia, dove l'area totale è suddivisa in "micro-aree" chiamate *patch*. In Alchemist, tuttavia, non è presente il concetto di "area" o "spazio", necessario per individuare una *patch*, in quanto le posizioni sono puntiformi e non necessariamente hanno coordinate intere. Il layer sviluppato ricalca l'area (rettangolare o quadrata) del sistema iniziale, implementando anche un sistema di conversione che trasforma la posizione puntiforme in modo tale da emulare la presenza di una "area" bidimensionale, a forma di quadrato, che corrisponde alla *patch*. Entrambe le misure, ovvero quella della griglia e quella della *patch* sono dinamiche e possono essere modificate dall'utente.

4.2 Le entità

Una volta definito l'ambiente di simulazione è importante determinare la struttura e la logica delle entità che lo abiteranno. Nella simulazione di riferimento 2.2 è possibile individuare come abitanti le tartarughe che, muovendosi, rilasciano una traccia chimica chiamata "feromone". Dunque, concettualmente, la tartaruga è un "contenitore" infinito di feromoni e, muovendosi, ne rilascia una parte nell'ambiente. Alchemist possiede il concetto di nodo 2.1.1 che, per definizione, è un contenitore di molecole che vive in un ambiente. È stato quindi necessario solamente definire la tipologia di molecola appartenente al nodo per tradurre questo aspetto della simulazione (ma anche del mondo reale) in Alchemist.

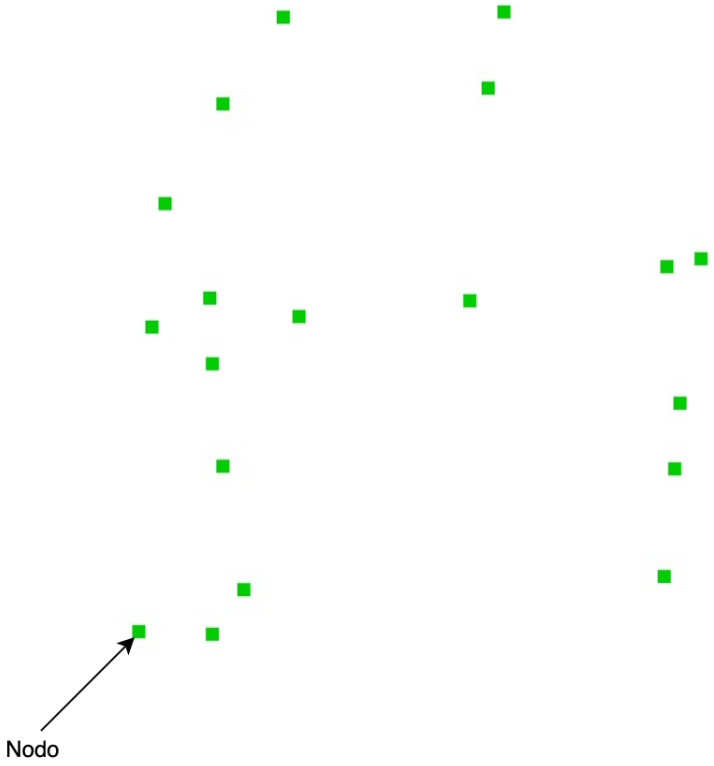

Figura 4.1: Rappresentazione grafica dove ogni punto è un nodo

4.3 Direzioni

Un aspetto importante da considerare è la direzione del movimento delle tartarughe. Nella simulazione di riferimento 2.2 le tartarughe si muovono e seguono un percorso che deriva dalla loro direzione. Essendo questo un attributo proprio delle tartarughe, è stato necessario sviluppare una soluzione che permettesse di inizializzare i nodi della simulazione con delle direzioni facilmente accessibili e modificabili in base alle esigenze.

Alchemist rende possibile la definizione di attributi personalizzati per i nodi, chiamati *Node Property*. Il corretto utilizzo di questa funzionalità ha permesso di definire un attributo di tipo *Direction* per ogni nodo che rappresentasse la direzione in modo tale da poter simulare in modo realistico questo aspetto della simulazione.

4.4 Reazioni globali

La Reazione Globale o *Global Reaction* rappresenta uno strumento fondamentale per descrivere le interazioni e i processi che avvengono all'interno di un sistema simulato. Possono essere utilizzate per modellare una vasta gamma di fenomeni, dalla diffusione chimica alla dinamica dei sistemi biologici. Contrariamente alle reazioni locali, le *Global Reaction* hanno effetti concreti sull'intero contesto in esame. Realizzare una modellazione precisa e realistica di questi fenomeni può essere di grande interesse per indagare tutti i risultati possibili.. Nell'ambito di questa tesi si individuano 3 tipi di reazioni, tutte collegate ai feromoni:

- Rilascio (*Deposit*)
- Evaporazione (*Evaporation*)
- Diffusione (*Diffusion*)

4.4.1 Rilascio

Questa reazione coinvolge in modo diretto il nodo, l'ambiente e il layer. Durante ogni movimento il nodo rilascia nell'ambiente una certa quantità di feromone in una

posizione discreta. Il layer si occupa della raccolta del feromone, posizionandolo in una *patch* ed incrementando il valore della sostanza collegato ad essa.

4.4.2 Evaporazione

L'evaporazione riguarda solamente il layer. In natura, con il passare del tempo, la traccia chimica diventa sempre più lieve fino a sparire completamente. Questa reazione si occupa esattamente di questo: ogni *patch* caratterizzata dalla presenza di un livello di feromone positivo viene individuata e il valore collegato ad essa viene decrementato in modo graduale.

4.4.3 Diffusione

La diffusione del feromone è un evento che caratterizza in modo diretto il layer e i nodi. Ogni volta che il nodo rilascia il feromone e questo si deposita in una *patch*, il layer diffonde nelle *patches* vicine delle tracce dello stesso.

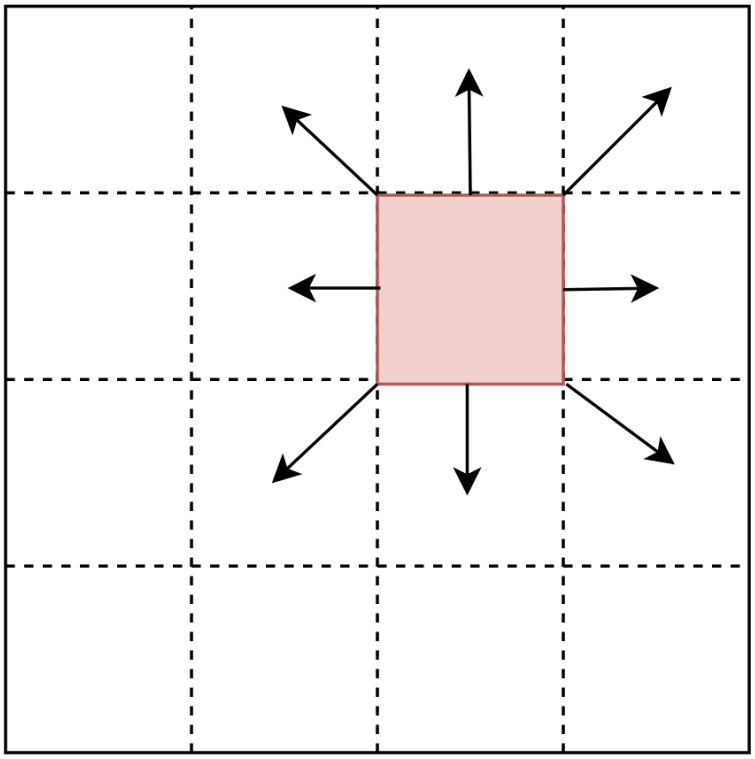

Figura 4.2: Diffusione grafica del feromone

Capitolo 5

Implementazione

5.1 Struttura del progetto

La struttura del progetto, organizzato in package, è la seguente:

- **Layer**: il layer personalizzato della simulazione.
- **Actions**: le azioni della simulazione.
- **GlobalReactions**: le azioni globali della simulazione.
- **NodeProperty**: le proprietà dei nodi della simulazione.

Per avviare il progetto in Alchemist, è necessario delineare accuratamente i parametri e le opzioni desiderate attraverso un file di configurazione YAML. Questo file fornisce le istruzioni necessarie per definire il comportamento del simulatore, specificare i componenti del sistema e regolare le interazioni tra di essi.

5.2 Layer

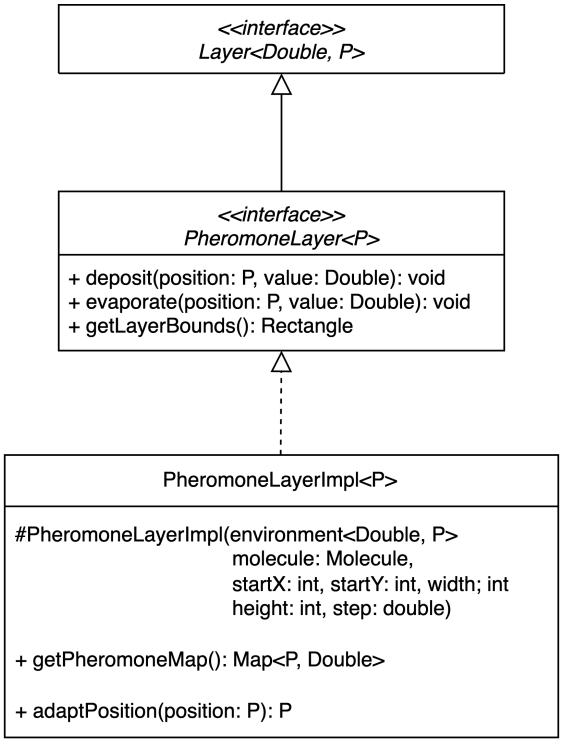

Figura 5.1: Struttura PheromoneLayer

Il `PheromoneLayer<P extends Position2D<P>>`, layer personalizzato della simulazione, è stato implementato come un'interfaccia che estende `Layer<T, P>` - interfaccia propria di Alchemist - dove `T` è il tipo di nodo e `P` è il tipo di posizione. L'utilizzo del parametro `P` implica che il `PheromoneLayer` può essere utilizzato con qualsiasi tipo di posizione, ma, nel contesto di questa tesi, si è preferito sfruttare posizioni `Position2D<P>` bidimensionali. Per la sua creazione è necessario definire 5 misure:

- `startX`: la coordinata x di partenza.
- `startY`: la coordinata y di partenza.
- `width`: la larghezza del layer.
- `height`: l'altezza del layer.

- `step`: la dimensione del passo, ovvero la lunghezza del lato di ogni *patch*.

Lo `step` è un parametro fondamentale per la corretta implementazione della simulazione in quanto Alchemist non possiede il concetto di area, necessaria per individuare una *patch*. Queste vengono rappresentate come "aree" puntiformi, e la loro dimensione (ovvero la distanza di un punto dall'altro) è appunto definita da questo parametro. Nella simulazione, il nodo deposita il feromone in una qualsiasi posizione, discreta e non obbligatoriamente intera, all'interno dei limiti dello spazio, e il `PheromoneLayer` si occupa di convertire questa posizione in una appartenente ad una *patch*. Il `PheromoneLayer` esegue, quindi, un arrotondamento per eccesso o per difetto, in modo tale da ottenere la posizione della *patch* più vicina.

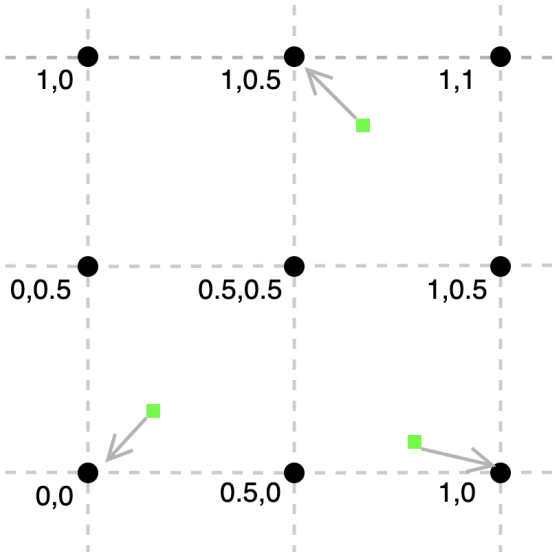

Figura 5.2: Rappresentazione grafica delle *patches* puntiformi. Ogni punto rappresenta una patch, mentre i quadratini rappresentano i nodi e la freccia indica su che *patch* il nodo depositerà il feromone. In questo esempio startX e startY hanno come valore 0, width e height 1 e step 0.5

5.2.1 Struttura dati

Un aspetto di fondamentale importanza riguarda la struttura dati utilizzata per la gestione del feromone. Per ovviare alla mancanza del concetto di area, è stato utilizzato un `HashMap<P, Double>` che associa ad ogni posizione `P` un valore

`Double` di feromone. Questa mappa viene inizializzata nel costruttore della classe attraverso il metodo `setupEnviromnent()` che si occupa di popolare la mappa con tutte le possibili posizioni delle *patch* e di inizializzare il feromone a 0.

```
1  public class PheromoneLayerImpl<P extends Position2D<P>> implements PheromoneLayer
       <P> {
2      ...
3      private final Map<P, Double> pheromoneMap;
4      ...
5      private void setupEnvironment(final int startX, final int startY){
6          for (final double x = startX; x <= width - Math.abs(startX); x = x + step)
               {
7              for (final double y = startY; y <= height - Math.abs(startY); y = y +
                   step){
8                  pheromoneMap.put(environment.makePosition(x, y), 0.0);
9              }
10         }
11     }
12     ...
13 }
```

5.2.2 Metodi

I metodi definiti nell'interfaccia e implementati nella classe sono:

- `void evaporate(P position, Double value)`: metodo che permette di far evaporare il feromone. Richiede in input la posizione e il valore del feromone.
- `void deposit(P position, Double value)`: metodo che permette di depositare il feromone. Richiede in input la posizione e il valore del feromone.
- `Rectangle getLayerBounds()`: metodo che restituisce un oggetto di tipo `Rectangle`, contentente i parametri per delineare l'area del Layer.

Di grande importanza sono i primi due metodi: `evaporate` e `deposit`. Entrambi sono nominati come le reazioni globali della simulazione e vengono utilizzati dalle stesse per accedere la mappa e modificare il feromone.

```java
public class PheromoneLayerImpl<P extends Position2D<P>> implements PheromoneLayer<P> {
    ...
    @Override
    public void deposit(final P p, final Double value){
        var mapPosition = adaptPosition(p);
        if(pheromoneMap.containsKey(mapPosition))
            pheromoneMap.put(mapPosition, (value + pheromoneMap.get(mapPosition)));
    }

    @Override
    public void evaporate(final P p, final Double value){
        if(pheromoneMap.containsKey(p))
            pheromoneMap.put(p, value>= 0 ? value : 0.0);
    }
    ...
}
```

5.3 Actions

In questa sezione vengono descritte le azioni della simulazione. Possiamo trovare:

- `MoveNode`: azione che permette ad ogni singolo nodo di muoversi.
- `NodeInfo`: azione che permette di controllare le informazioni di ogni singolo nodo.

5.3.1 MoveNode

La classe `MoveNode<P extends Position<P> & Position2D<P>>` incorpora l'intera logica che permette il movimento di ogni singolo nodo. Per la sua creazione è necessario che l'utente definisca i seguenti parametri:

- `sniffThreshold`: la soglia di feromone che il nodo deve percepire per potersi muovere.
- `sniffDistance`: la distanza del passo di movimento.
- `wiggleBias`: la tendenza a preferire un movimento casuale oscillatorio in una direzione specifica.

Il parametro `wiggleBias` può essere impostato in 3 modi:

- `0`: il nodo ha il 50% di muoversi in avanti e il 25% di muoversi a destra o a sinistra.
- `0 > x <= 40`: il nodo tende a preferire la direzione di sinistra; il valore indica la probabilità di muoversi in quel verso. Il valore 40 indica il 100% di probabilità di muoversi in quella direzione.
- `-40 => x < 0`: il nodo tende a preferire la direzione di destra; il valore indica la probabilità di muoversi in quel verso. Il valore -40 indica il 100% di probabilità di muoversi in quella direzione.

La classe `MoveNode` estende la classe astratta `AbstractAction<T>`, facente parte del set base di Alchemist, implementandone i metodi astratti. Tra questi, il metodo `execute` è il più importante, in quanto si occupa di eseguire l'azione di movimento vera e propria. La logica di movimento segue questi passi:

1. Viene individuata la posizione attuale del nodo.
2. Questa posizione viene adattata alla *patch* più vicina.
3. Vengono calcolate le direzioni possibili in base alle patch adiacenti alla posizione calcolata precedentemente che hanno una concentrazione di feromone superiore ad una soglia definita dall'utente (il parametro si chiama `sniffThreshold`).
4. Viene identificata la *patch* con la maggiore concentrazione di feromone. Tuttavia, questa, non è sempre individuabile. Vi sono due casi possibili: nel primo, nella *patch* è presente un valore di feromone, ma esso è sotto la soglia minima `sniffThreshold` e dunque la *patch* viene scartata; nel secondo caso, invece, nella *patch* non è presente alcuna traccia di feromone, e dunque questa non viene proprio rilevata.
5. Se la *patch* è presente:
   (a) La direzione del nodo viene aggiornata in base alla direzione della *patch* con la maggiore concentrazione di feromone.

(b) Il nodo si muove verso quella *patch* e si aggiorna la direzione del nodo.

6. Se la *patch* non è presente:

   (a) Viene calcolata una direzione casuale tra quelle possibili (dritto, verso destra o verso sinistra a seconda del valore del parametro `wiggleBias`), tenendo in considerazione la direzione attuale del nodo.

   (b) Il nodo si muove nella direzione scelta e l'orientamento del nodo viene aggiornato.

5.3.2 NodeInfo

La classe `NodeInfo<T, P extends Position<P> & Position2D<P>>` permette di tracciare le informazioni di ogni singolo nodo. Anche questa classe estende la classe astratta `AbstractAction<T>`, implementandone i metodi. Le informazioni osservabili sono le seguenti:

- `PheromoneValue`: la concentrazione di feromone nella *patch* in cui si trova il nodo.
- `direction`: la direzione attuale del nodo.
- `pheromone`: la quantità di feromone che il nodo rilascia.

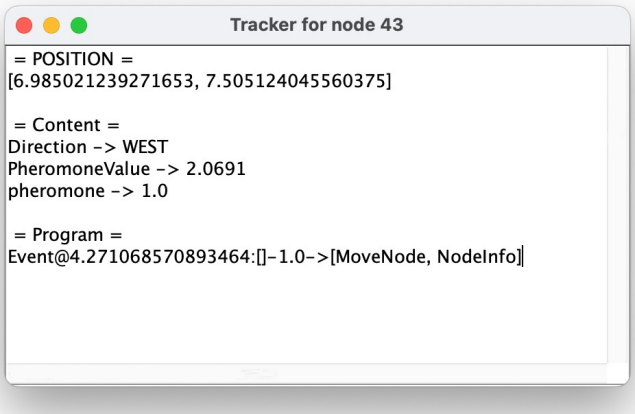

Figura 5.3: Le informazioni del nodo. Si possono osservare nella sezione “Content”

5.4 GlobalReactions

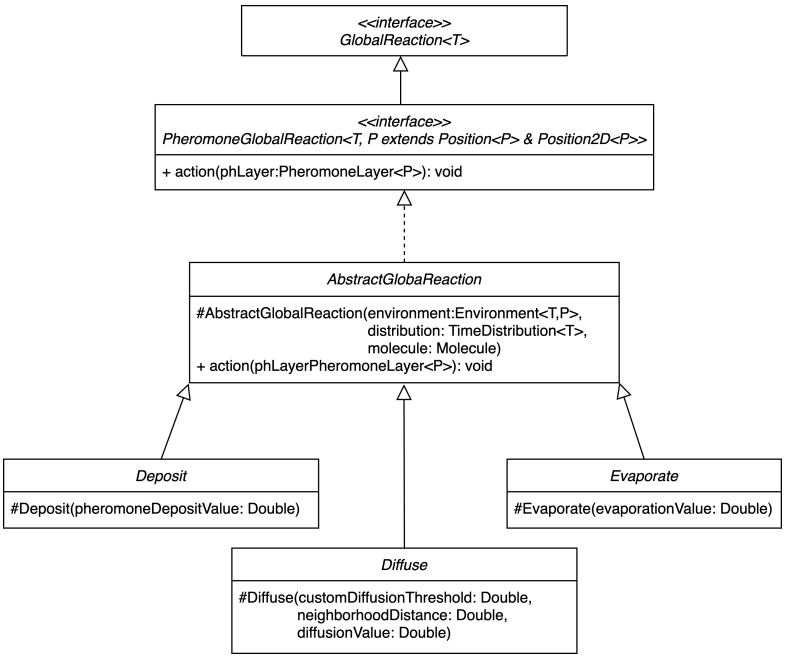

Figura 5.4: Schema delle Global Reaction

In questa sezione del progetto sono state implementate le reazioni globali della simulazione. Possiamo trovare:

- `Evaporate`: azione che permette di far evaporare il feromone.
- `Deposit`: azione che permette di far diffondere il feromone.
- `Diffuse`: azione che diffonde il feromone nelle *patch* adiacenti.

Le classi sopra introdotte estendono la classe astratta `AbstractGlobalReaction<T, P extends Position<P> & Position2D<P>` la quale implementa l'interfaccia `PheromoneGlobalReaction<T, P>`. Quest'ultima estende l'interfaccia propria di Alchemist `GlobalReaction<T>` dove sono definite le operazioni da implementare in modo tale che il simulatore possa identificare ed utilizzare in modo corretto tutti i sorgenti necessari. L'interfaccia `PheromoneGlobalReaction<T, P>` definisce il metodo `action(PheromoneLayerImpl<P> phLayer)`. Quest'ultimo è implementato in ogni classe e si occupa di eseguire la logica della reazione. Si è scelta

questa struttura in quanto i metodi definiti dall'interfaccia `GlobalReaction<T>` sono i medesimi per ogni azione globale e quindi l'utilizzo della classe astratta `AbstractGlobalReaction`, che li implementa, permette di evitare la ripetizione inutile del codice.

La simulazione esegue una volta al secondo le reazioni in modo tale da garantire un'evoluzione corretta del feromone nel tempo. Inoltre, le reazioni in questione vengono eseguite nell'ordine in cui sono state definite nel file di configurazione YAML.

5.4.1 Evaporate

Questa classe ha il compito di fare evaporare il feromone dall'ambiente. L'azione di evaporazione è simulata attraverso la moltiplicazione del valore del feromone per un coefficente di evaporazione compreso tra 0 e 1, definito dall'utente. Questo evento ha effetto su ogni singola *patch* in cui è presente il feromone. Per alterare il valore del feromone di ogni *patch*, viene richiamato il metodo `evaporate` del `PheromoneLayer` 5.2.

```
1  public class Evaporate<T, P extends Position<P> & Position2D<P>> extends
       AbstractGlobalReaction<T, P> {
2      ...
3      @Override
4      public void action(final PheromoneLayerImpl<P> phLayer) {
5          Map<P, Double> pheromoneMap = phLayer.getPheromoneMap();
6          pheromoneMap.forEach((key, value) -> {
7              if (value > 0) {
8                  phLayer.evaporate(key, value * evaporationValue);
9              }
10         });
11     }
12     ...
13 }
```

5.4.2 Deposit

Il sorgente protagonista di questa sotto-sezione compie l'azione di depositare il feromone. Dall'ambiente vengono individuate le posizioni di tutti i nodi e viene poi richiamato il metodo `deposit` del `PheromoneLayer` 5.2 per modificare il valore del

feromone, associato alla posizione attuale del nodo, presente nella struttura dati 5.2.1.

```
public class Deposit<T, P extends Position<P> & Position2D<P>> extends
    AbstractGlobalReaction<T, P> {
    ...
    @Override
    public void action(final PheromoneLayerImpl<P> phLayer) {
        final Environment<T, P> environment = this.getEnvironment();

        for (var node : this.nodeList) {
            P pos = environment.getPosition(node);
            phLayer.deposit(pos, pheromoneDepositValue);
        }
    }
    ...
}
```

5.4.3 Diffuse

Quest'ultima classe si occupa di diffondere il feromone nelle *patch* adiacenti. Vengono quindi individuate tutte le *patch*, e per ognuna si calcola il suo vicinato. Se la *patch* ha un valore di feromone superiore ad una soglia definita dall'utente, si procede a diffondere il feromone. La diffusione è emulata attraverso la moltiplicazione del valore del feromone per un coefficente di diffusione deciso dall'utente. Questa reazione, concettualmente, è simile alla `Deposit` in quanto nel vicinato di una *patch* viene depositato del feromone, e per questo motivo, viene richiamato il metodo `deposit` del `PheromoneLayer` 5.2.

```
public class Diffuse<T, P extends Position<P> & Position2D<P>> extends
    AbstractGlobalReaction<T, P> {
    ...
    @Override
    public void action(final PheromoneLayerImpl<P> phLayer) {
        var pheromoneMap = phLayer.getPheromoneMap();
        pheromoneMap.forEach((k, v) -> {
            if (v > customDiffusionThreshold){
                getNeighborhood(k).forEach(x -> phLayer.deposit(x, v *
                    diffusionValue));
            }
        });
```

```
11      }
12      ...
13  }
```

5.5 NodeProperty

La classe `DirectionProperty` è stata implementata per poter associare ad ogni nodo una proprietà che rispecchiasse la sua direzione attuale. Estende la classe `AbstractNodeProperty<T>` fornita dal set base di Alchemist. L'enum `Direction`, che definisce le direzioni possibili in cui un nodo può muoversi, è propedeutico all'utilizzo di questa classe. L'enum definisce i 4 punti cardinali e i loro punti intermedi. Infatti, un nodo può muoversi in 8 direzioni diverse e, tenere traccia di queste, è fondamentale per la corretta esecuzione della simulazione. L'enum, oltre a contenere le definizioni delle direzioni, contiene anche, per ognuna di esse, metodi che ne ritornano le coordinate x e y e le direzioni adiacenti. All'avvio del programma, ad ogni nodo viene assegnata una direzione in modo randomico.

La `NodeProperty` è una proprietà fondamentale per il corretto funzionamento dell'azione `MoveNode` 5.3.1; senza di essa, infatti, il movimento del nodo risulterebbe irrealistico in quanto si muoverebbe in modo completamente casuale. Con l'implementazione di questa classe e una logica di movimento (implementata nella classe `MoveNode` 5.3.1) pensata per sfruttare questa componente, invece, si osserva che il movimento di un singolo nodo risulta piuttosto "armonioso" e realistico, più simile a quello di un essere vivente che esplora l'ambiente circostante.

Capitolo 6

Evaluation

Capitolo 7

Conclusioni e ringraziamenti

Bibliografia

[GB00] MA Gibson and J Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.

[Gil77] DT Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[PMV13] D Pianini, S Montagna, and M Viroli. Chemical-oriented simulation of computational systems with alchemist. *Journal of Simulation*, 7(3):202–215, August 2013.

[Wil97] Uri Wilensky. Netlogo slime model. `http://ccl.northwestern.edu/netlogo/models/Slime`, 1997.

[Wil99] Uri Wilensky. Netlogo. `http://ccl.northwestern.edu/netlogo/`, 1999.