

Corso di Laurea in Ingegneria e Scienze Informatiche

Simulazione di fenomeni emergenti in Alchemist: il caso dell'aggregazione di *slime-mold*

Tesi di laurea in:
PROGRAMMAZIONE AD OGGETTI

Relatore

Prof. Mirko Viroli

Candidato

Lorenzo Tosi

Correlatore

Dott. Gianluca Aguzzi

Abstract

I comportamenti emergenti rappresentano il manifestarsi di fenomeni collettivi che sorgono dall'interazione di molteplici componenti di un sistema. Si possono osservare in diversi ambiti e, per questo, sono diventati oggetto di interesse e studio da parte della comunità scientifica. La caratteristica principale di questi comportamenti è la loro imprevedibilità in quanto non possono essere dedotti dalle leggi che regolano il comportamento del singolo.

Lo scopo di questa tesi è l'esplorazione di questi fenomeni attraverso lo sviluppo di un sistema software che si interfacci con il simulatore Alchemist e che permetta di emulare il fenomeno di aggregazione di singoli organismi in gruppi. Questo documento fa una panoramica sul contesto scientifico in cui si colloca il lavoro svolto, analizza i requisiti e le specifiche del sistema da sviluppare, ne illustra il design, ne presenta l'implementazione e discute i risultati ottenuti dalle simulazioni effettuate.

Optional. Max a few lines.

Acknowledgements

Optional. Max 1 page.

Indice

Abstract	iii
1 Introduzione	1
2 Contesto	5
2.1 NetLogo	5
2.1.1 Simulazione di riferimento	6
2.2 Overview su <i>slime-mold</i>	7
2.3 Alchemist	8
2.3.1 Il meta-modello	9
3 Analisi del dominio	11
3.1 Dominio	11
3.2 Requisiti	11
4 Design	13
4.1 Layer	13
4.1.1 Le posizioni	14
4.2 I nodi	15
4.3 Reazioni globali	16
4.3.1 Rilascio	17
4.3.2 Evaporazione	17
4.3.3 Diffusione	17
4.4 Direzioni	18
5 Implementazione	19
5.1 Struttura del progetto	19
5.2 Layer	19
5.2.1 Struttura dati	21
5.2.2 Metodi	21
5.3 Actions	22

5.3.1	MoveNode	22
5.3.2	NodeInfo	24
5.4	GlobalReactions	26
5.4.1	Evaporate	26
5.4.2	Deposit	27
5.4.3	Diffuse	28
5.5	NodeProperty	28
6	Evaluation	31
6.1	Simulazione 1	31
6.2	Simulazione 2	32
6.3	Simulazione 3	33
6.4	Simulazione 4	33
6.5	Simulazione 5	34
6.6	Risultati delle simulazioni	35
7	Conclusioni e lavori futuri	39
7.1	Lavori futuri	39
8	Ringraziamenti	41
		43
	Bibliografia	43

Elenco delle figure

2.1	Simulazione di riferimento	6
2.2	Il mondo suddiviso in patch	7
2.3	(a) Reazione in Alchemist (b) Il modello di alchemist	10
3.1	Schema geneale del dominio.	12
4.1	Struttura PheromoneLayer	14
4.2	Rappresentazione grafica dove ogni punto è un nodo.	15
4.3	Schema delle Global Reaction	16
4.4	Il processo di diffusione; la figura (a) mostra il feromone in una <i>patch</i> , la figura (b) mostra le <i>patch</i> vicine nelle quali il feromone verrà diffuso e la figura (c) mostra lo stato dell'ambiente una volta che la diffusione è terminata.	17
5.1	Rappresentazione grafica delle <i>patches</i> puntiformi. Ogni punto rappresenta una patch, mentre i quadratini rappresentano i nodi e la freccia indica su che <i>patch</i> il nodo depositerà il feromone. In questo esempio startX e startY hanno come valore 0, width e height 1 e step 0.5	20
5.2	Le informazioni del nodo. Si possono osservare nella sezione "Content"	25
6.1	(a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi	35
6.2	(a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi	35
6.3	(a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi	36
6.4	(a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi	36
6.5	(a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi	37

List of Listings

listings/phlayerSetup.java	21
listings/phLayerDepositAndEvaporate.java	22
listings/evaporate.java	27
listings/deposit.java	27
listings/diffuse.java	28

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.

A livello informatico e tecnologico si possono trovare molti esempi di comportamenti emergenti in quanto, ad oggi, i sistemi informatici hanno raggiunto una complessità tale da poter manifestare comportamenti simili a quelli osservabili in natura. Un primo esempio di comportamento si può trovare nell'*IoT Energy Management*. In questo caso vengono sfruttate le tecnologie dell'Internet of Things (IoT) per monitorare e gestire il consumo energetico di una zona. Ogni dispositivo è programmato per reagire a stimoli provenienti da altri dispositivi e, al contempo, lascia segnali agli altri dispositivi. Questo comportamento collettivo permette di raccogliere dati utili per ottimizzare il consumo energetico dell'intera zona. Più interessante a livello di questa tesi è, invece, il fenomeno della *swarm robotics* o “robotica degli sciami”. Questo campo di ricerca fonde la robotica con l'intelligenza artificiale prendendo come esempio i comportamenti degli insetti sociali in modo tale da poterli studiare e riprodurre. Ogni robot è programmato per manifestare lo stesso comportamento che un insetto manifesterebbe in natura, sia come individuo che come collettivo. Questi studi hanno permesso di comprendere che l'intelligenza manifestata dal collettivo supera di gran lunga quella del singolo individuo, al punto che si parla di intelligenza collettiva o “swarm intelligence”.

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

- Comprimerne 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. Attualmente Alchemist non supporta nativamente la possibilità di simulare di fenomeni emergenti. Attraverso questa tesi si sono introdotte le astrazioni necessarie per modellarli, prendendo come riferimento il caso della *slime-mold aggregation*.

La tesi presenta la seguente struttura:

- **Contesto:** in questo capitolo si introduce il contesto scientifico in cui si colloca il lavoro svolto, presentando le tecnologie adottate e la simulazione di riferimento.
- **Analisi:** in questo capitolo si analizzano i requisiti e le specifiche del sistema da sviluppare.
- **Design:** in questo capitolo si illustra il design del sistema, presentando le scelte progettuali e le motivazioni che hanno portato a queste.
- **Implementazione:** in questo capitolo si illustra l'implementazione del sistema, presentando le scelte implementative e mostrando parti di codice significative.
- **Evaluation:** in questo capitolo si discutono i risultati ottenuti dalle simulazioni effettuate.
- **Conclusioni e lavori futuri** in questo capitolo si presentano le conclusioni del lavoro svolto e si discutono possibili sviluppi futuri.
- **Ringraziamenti:** in questo capitolo si ringraziano le persone che hanno contribuito alla realizzazione di questa tesi.

Capitolo 2

Contesto

In questo capitolo viene spiegato il contesto scientifico in cui si colloca il lavoro svolto. Nello stato dell'arte attuale sono presenti diversi simulatori che permettono di emulare comportamenti emergenti, come ad esempio **NetLogo**. Questo strumento ha rivestito un'importanza fondamentale per un riferimento iniziale e per la comprensione del fenomeno in esame. In particolare, la **simulazione di riferimento** “Slime” è stata utilizzata come base per lo sviluppo del modello di simulazione in **Alchemist**. A partire da ciò, nasce l'interesse di indagare un altro fenomeno dalle caratteristiche estremamente simili: la **muffa mucillaginosa**. Questo lavoro di tesi si propone con l'obiettivo di sviluppare le principali astrazioni, necessarie per modellare il questo fenomeno, all'interno del simulatore **Alchemist**.

2.1 NetLogo



NetLogo¹ [Wil97] è un ambiente di programmazione e simulazione open-source progettato per eseguire simulazioni di modelli complessi e dinamici. È stato sviluppato da Uri Wilensky e collaboratori presso il Center for Connected Learning and Computer-Based Modeling presso la Northwestern University. Il linguaggio di programmazione di NetLogo è molto semplice e permette di scrivere codice in modo intui-

¹<https://ccl.northwestern.edu/netlogo/>

tivo e veloce. Inoltre, NetLogo è dotato di un'interfaccia grafica che permette di visualizzare in tempo reale il comportamento del sistema simulato.

NetLogo è un simulatore ad agenti, ovvero un programma che simula il comportamento di un insieme di agenti che interagiscono tra di loro e con l'ambiente circostante. Ogni agente in NetLogo può rappresentare una vasta gamma di entità con propria autonomia decisionale. Gli agenti più comuni sono le *tartarughe*, ovvero entità “vive” che si muovono in uno spazio, *patch*, ovvero le celle che compongono lo spazio, i *link*, ovvero le connessioni tra le *tartarughe* e le *patch* e gli *observer*, ovvero l'agente monitor della simulazione.

2.1.1 Simulazione di riferimento

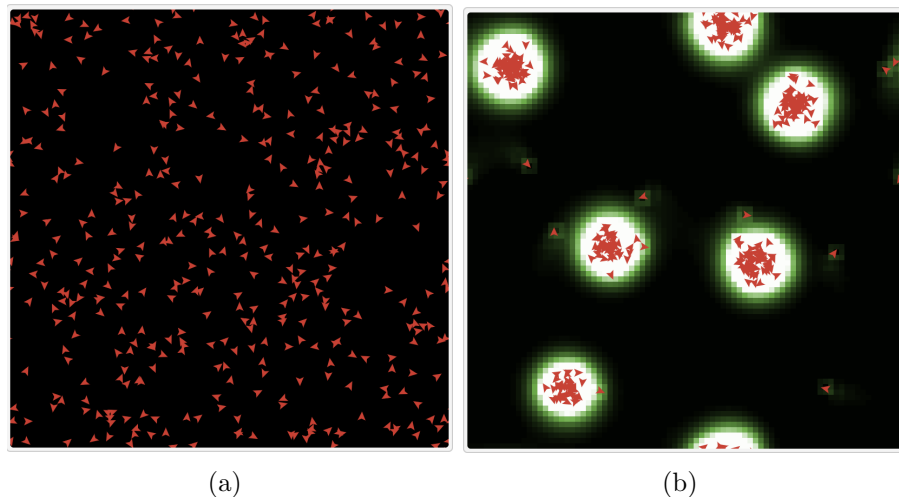


Figura 2.1: Simulazione di riferimento

La simulazione² utilizzata come riferimento per questo progetto di tesi è presente nella libreria di modelli di NetLogo. Il modello di riferimento è “Slime” [Wil97] fig. 2.1. e per simulare l’aggregazione di tanti singoli organismi in un gruppo vengono usati gli agenti sopra descritti, ovvero le *patch* e le *tartarughe*. Quest’ultime si muovono in uno spazio a griglia e durante il loro movimento rilasciano una

²<https://www.netlogoweb.org/launch#http://www.netlogoweb.org/assets/modelslib/Sample%20Models/Biology/Slime.nlogo>

particolare molecola chiamata “feromone” che si deposita in una posizione precisa. L’intero mondo è quindi suddiviso in tantissime “micro-aree” chiamate *patch* fig. 2.2. 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.

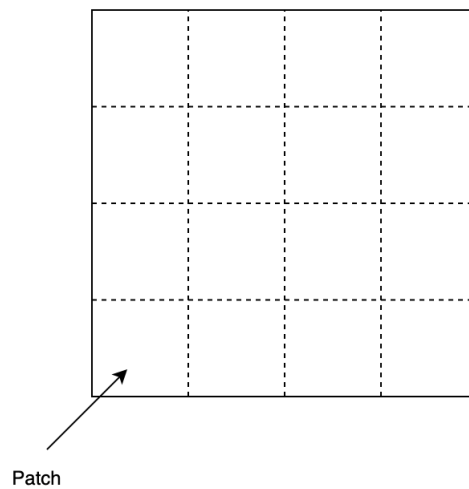


Figura 2.2: Il mondo suddiviso in patch

2.2 Overview su *slime-mold*

Nel contesto della presente tesi, sulla base della simulazione di riferimento 2.1.1 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.

2.3 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

- Scafì

2.3.1 Il meta-modello

Come accennato in precedenza, il meta-modello fig. 2.3 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 fig. 2.3. 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 dell’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.

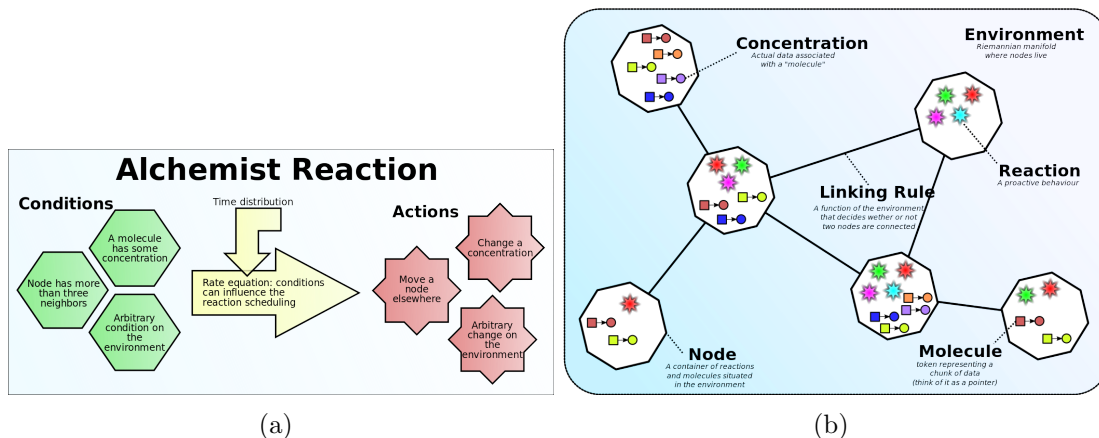


Figura 2.3: (a) Reazione in Alchemist (b) Il modello di alchemist

Capitolo 3

Analisi del dominio

3.1 Dominio

Il dominio di interesse è caratterizzato dalla presenza di entità in un ambiente. Queste entità sono vive e si muovono. Durante il loro movimento rilasciano una traccia, sotto forma di sostanza chimica, simile ad un odore che può essere percepito dalle altre entità presenti nell'ambiente. Questa sostanza si deposita in un punto dell'ambiente ed inizia ad espandersi. Con il passare del tempo la sostanza, evaporando, svanisce dall'ambiente. Quando un'entità, muovendosi, percepisce la presenza di questa sostanza, viene influenzata a muoversi verso la direzione in cui l'ha percepita. Questo comportamento, riprodotto in larga scala, porta alla formazione di strutture complesse e funzionali. Un esempio di questo comportamento è osservabile in natura, nel caso delle formiche: queste, infatti, riescono a creare disposizioni complesse e funzionali per tracciare il cibo e per costruire i loro nidi.

3.2 Requisiti

Analizzando il dominio possiamo individuare e sintetizzare le caratteristiche e i requisiti che la simulazione 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.

Segue uno schema generale del dominio, come rappresentato in figura 3.1.

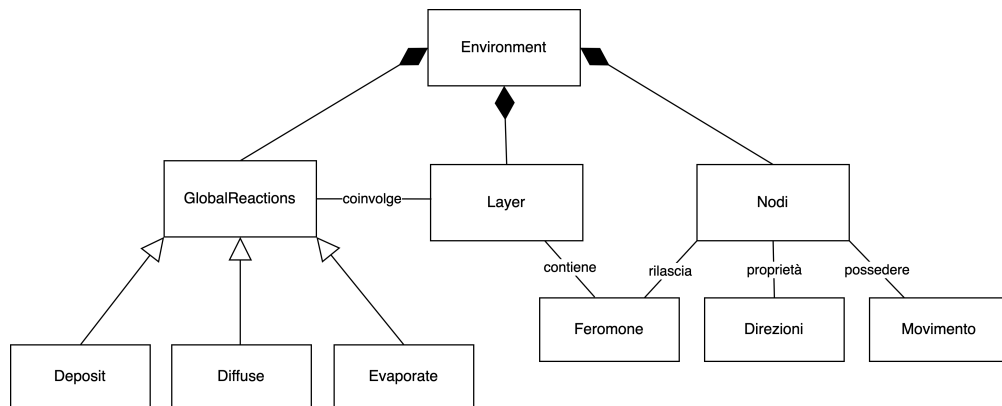


Figura 3.1: Schema geneale del dominio.

Non sono stati individuati ulteriori requisiti specifici per la realizzazione di questo progetto di tesi. Lo scopo di questa ricerca, infatti, è stato soprattutto quello di comprendere come il simulatore Alchemist potesse essere in grado di simulare questi comportamenti, tramite lo sviluppo un modello dimostrativo.

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 2.1.1, 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*.

Di seguito si riporta un diagramma UML fig. 4.1 che rappresenta il layer.

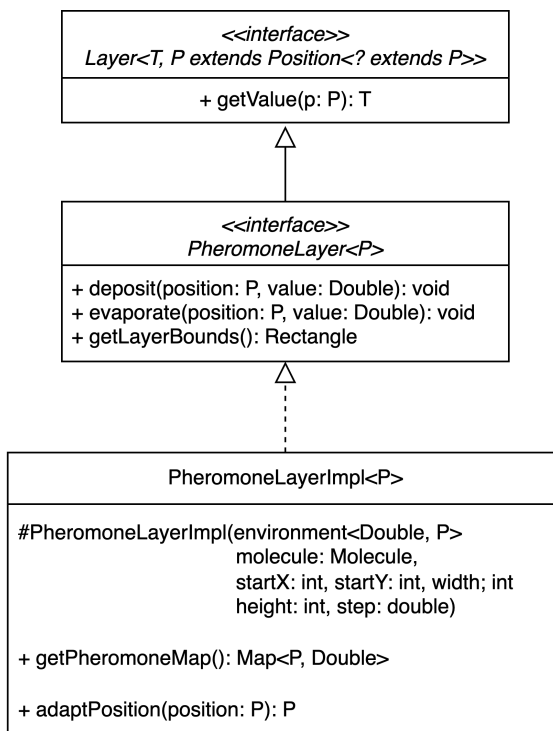


Figura 4.1: Struttura PheromoneLayer

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 2.1.1 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 I nodi

Una volta definito l'ambiente di simulazione è importante determinare la struttura e la logica delle entità che lo abiteranno. Nella simulazione di riferimento 2.1.1 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 2.3.1 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.

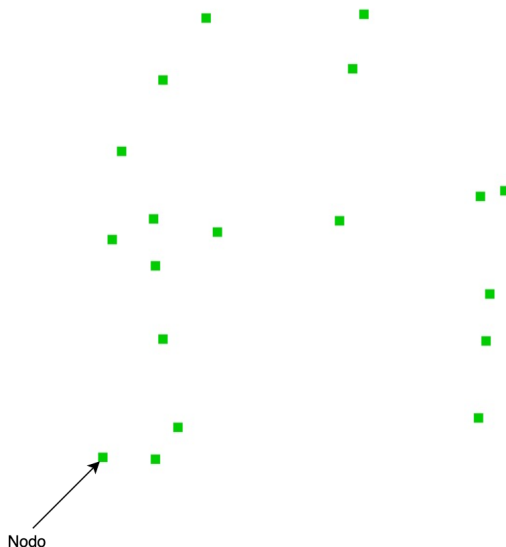


Figura 4.2: Rappresentazione grafica dove ogni punto è un nodo.

4.3 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.

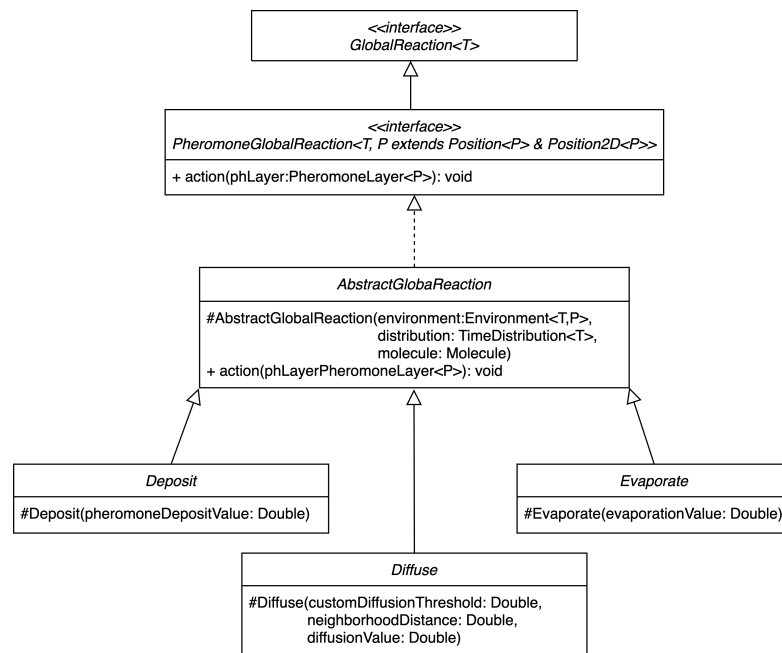


Figura 4.3: Schema delle Global Reaction

Come si vede in figura fig. 4.3, nell'ambito di questa tesi si individuano 3 tipi di reazioni, tutte collegate ai feromoni:

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

4.3.1 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.

4.3.2 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.

4.3.3 Diffusione

La diffusione fig. 4.4 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.

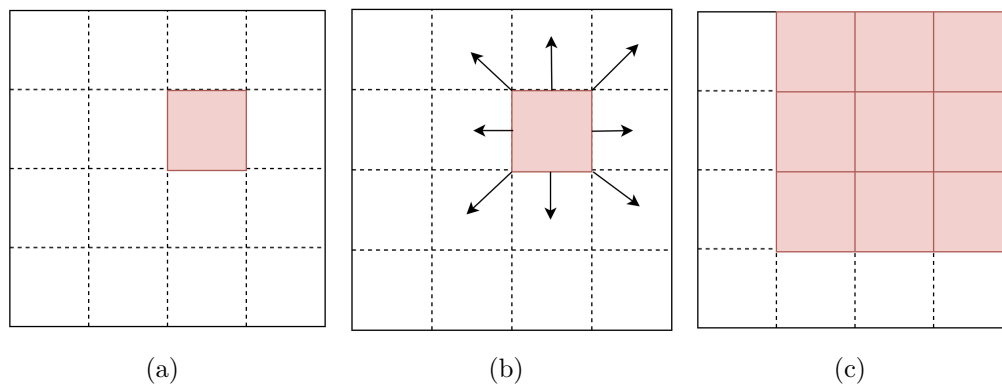


Figura 4.4: Il processo di diffusione; la figura (a) mostra il feromone in una *patch*, la figura (b) mostra le *patch* vicine nelle quali il feromone verrà diffuso e la figura (c) mostra lo stato dell'ambiente una volta che la diffusione è terminata.

4.4 Direzioni

Un aspetto importante da considerare è la direzione del movimento delle *tartarughe*. Nella simulazione di riferimento 2.1.1 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.

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

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.

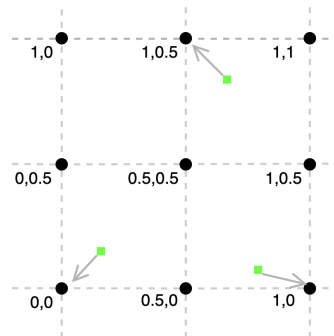


Figura 5.1: Rappresentazione grafica delle *patches* puntiformi. Ogni punto rappresenta una *patch*, mentre i quadratini rappresentano i nodi e la freccia indica su che *patch* il nodo depositerà il feromone. In questo esempio **startX** e **startY** hanno come valore 0, **width** e **height** 1 e **step** 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 `setupEnvironment()` 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         {
8             for (final double y = startY; y <= height - Math.abs(startY); y = y +
               step){
9                 pheromoneMap.put(environment.makePosition(x, y), 0.0);
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`, contenente 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.

```
1 public class PheromoneLayerImpl<P extends Position2D<P>> implements PheromoneLayer
2     <P> {
3     ...
4     @Override
5     public void deposit(final P p, final Double value){
6         var mapPosition = adaptPosition(p);
7         if(pheromoneMap.containsKey(mapPosition))
8             pheromoneMap.put(mapPosition, (value + pheromoneMap.get(mapPosition)))
9         ;
10    }
11
12    @Override
13    public void evaporate(final P p, final Double value){
14        if(pheromoneMap.containsKey(p))
15            pheromoneMap.put(p, value>= 0 ? value : 0.0);
16    }
17    ...
18 }
```

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.
- $0 > x \leq 40$: 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.
- $-40 \Rightarrow x < 0$: 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, come raffigurato in figura fig. 5.2. 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.

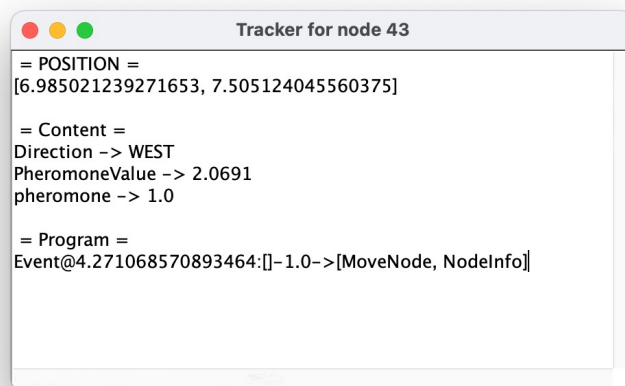


Figura 5.2: Le informazioni del nodo. Si possono osservare nella sezione “Content”

5.4 GlobalReactions

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 coefficiente 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 }
```

[ht]

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.

```
1 public class Deposit<T, P extends Position<P> & Position2D<P>> extends
  AbstractGlobalReaction<T, P> {
2     ...
3     @Override
4     public void action(final PheromoneLayerImpl<P> phLayer) {
5         final Environment<T, P> environment = this.getEnvironment();
6
7         for (var node : this.nodeList) {
8             P pos = environment.getPosition(node);
9             phLayer.deposit(pos, pheromoneDepositValue);
10        }
11    }
12    ...
13 }
```

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 coefficiente 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.

```
1 public class Diffuse<T, P extends Position<P> & Position2D<P>> extends
    AbstractGlobalReaction<T, P> {
2     ...
3     @Override
4     public void action(final PheromoneLayerImpl<P> phLayer) {
5         var pheromoneMap = phLayer.getPheromoneMap();
6         pheromoneMap.forEach((k, v) -> {
7             if (v > customDiffusionThreshold){
8                 getNeighborhood(k).forEach(x -> phLayer.deposit(x, v *
                    diffusionValue));
9             }
10        });
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

In questa sezione vengono presentati i risultati delle simulazioni effettuate e i loro valori. Ogni simulazione presenta la propria immagine che ne dimostra i risultati nella sezione *Risultati delle simulazioni* sezione 6.6.

6.1 Simulazione 1

I valori di questa simulazione fig. 6.1 sono i seguenti:

Tabella 6.1: Valori dei parametri

Parametro	Valore
Numero di nodi	500
sniffThreshold	1.5
wiggleBias	0
evaporation	0.6
diffusion	0.5
deposit	1
startX	-15
startY	-15
width	30
height	30
step	0.5
customDiffusionTreshold	5

Dalle dimensioni dell'ambiente si può osservare che il numero di *patch* è pari a

3721, quindi 3721 punti che immagazzineranno la quantità di feromone depositata dai nodi. Ci si aspetta che l'aggregazione avvenga abbastanza rapidamente e in zone definite data l'alta densità di punti di raccolta del feromone e, in generale, dell'ambiente. Con 500 nodi ci si aspetta che appena dei nodi sono vicini rimangano "intrappolati" nelle loro posizioni, depositando il feromone e generando un piccolo "nucleo" di aggregazione. Una volta formato, tutti i nodi che si trovano nelle vicinanze saranno attratti e contribuiranno a rafforzare l'aggregazione.

6.2 Simulazione 2

I valori di questa simulazione fig. 6.2 sono i seguenti:

Tabella 6.2: Valori dei parametri	
Parametro	Valore
Numero di nodi	500
<code>sniffThreshold</code>	4
<code>wiggleBias</code>	0
<code>evaporation</code>	0.6
<code>diffusion</code>	$\frac{1}{18}$
<code>deposit</code>	1
<code>startX</code>	-15
<code>startY</code>	-15
<code>width</code>	30
<code>height</code>	30
<code>step</code>	0.5
<code>customDiffusionTreshold</code>	1

Questa simulazione presenta gli stessi valori della precedente, ma con un valore di `sniffThreshold` più alto e un valore di `diffusion` più basso. Ci si aspetta che l'aggregazione avvenga più lentamente, i nodi si muovano per più tempo nello spazio prima di aggregarsi o di trovare un agglomerato già presente e che, generamnte, ci saranno meno zone di aggregazione: quindi quelle che ci saranno conterranno un grande numero di nodi. Il motivo di queste deduzioni è principalmente legato al fatto che il valore di `diffusion` è più basso rispetto alla simulazione precedente sezione 6.1 e il valore di `sniffThreshold` è più alto. Sarà necessario che un maggior numero di nodi si incontri in un punto per formare un cluster.

6.3 Simulazione 3

I valori di questa simulazione fig. 6.3 sono i seguenti:

Tabella 6.3: Valori dei parametri

Parametro	Valore
Numero di nodi	100
<code>sniffThreshold</code>	1.5
<code>wiggleBias</code>	0
<code>evaporation</code>	0.6
<code>diffusion</code>	0.5
<code>deposit</code>	1
<code>startX</code>	-15
<code>startY</code>	-15
<code>width</code>	30
<code>height</code>	30
<code>step</code>	0.5
<code>customDiffusionTreshold</code>	5

Questa simulazione dimostra che, pur con un numero di nodi estremamente inferiore rispetto alle precedenti, l'aggregazione avviene comunque. I valori sono gli stessi della prima simulazione sezione 6.1, ma con un numero di nodi pari a 100. Si osserva che dopo 100 secondi si inizia a sviluppare un'aggregazione lieve, in pochissimi punti dello spazio. Dopo 300 secondi si può notare che l'aggregazione si è rafforzata in quei punti, ma sono presenti ancora diversi nodi vaganti.

6.4 Simulazione 4

Questa simulazione presenta gli stessi valori della seconda simulazione sezione 6.2, ma con un numero di nodi pari a 100. Lo scopo di questa simulazione è quello di confrontare i risultati con la terza simulazione sezione 6.3, che presenta gli stessi valori della prima simulazione sezione 6.1, ma con differenze nel `sniffThreshold` e `diffusion`. In questo caso non avviene alcuna aggregazione e il motivo principale è legato alle differenze di valori di `sniffThreshold` e `diffusion`.

I valori di questa simulazione fig. 6.4 sono i seguenti:

Tabella 6.4: Valori dei parametri

Parametro	Valore
Numero di nodi	100
sniffThreshold	4
wiggleBias	0
evaporation	0.6
diffusion	$\frac{1}{18}$
deposit	1
startX	-15
startY	-15
width	30
height	30
step	0.5
customDiffusionTreshold	1

6.5 Simulazione 5

I valori di questa simulazione fig. 6.5 sono i seguenti:

Tabella 6.5: Valori dei parametri

Parametro	Valore
Numero di nodi	500
sniffThreshold	4
wiggleBias	0
evaporation	0.5
diffusion	1
deposit	2
startX	-10
startY	-10
width	20
height	20
step	0.5
customDiffusionTreshold	10

Quest'ultima simulazione presenta valori diversi da quelle precedenti. Si è voluto indagare il caso in cui si avesse un ambiente più piccolo e dei valori di **evaporation**, **diffusion** e **deposit** più alti. Con questa configurazione si osserva una quasi

immediata aggregazione dei nodi in tanti punti: questo comportamento era atteso dati i valori di partenza.

6.6 Risultati delle simulazioni

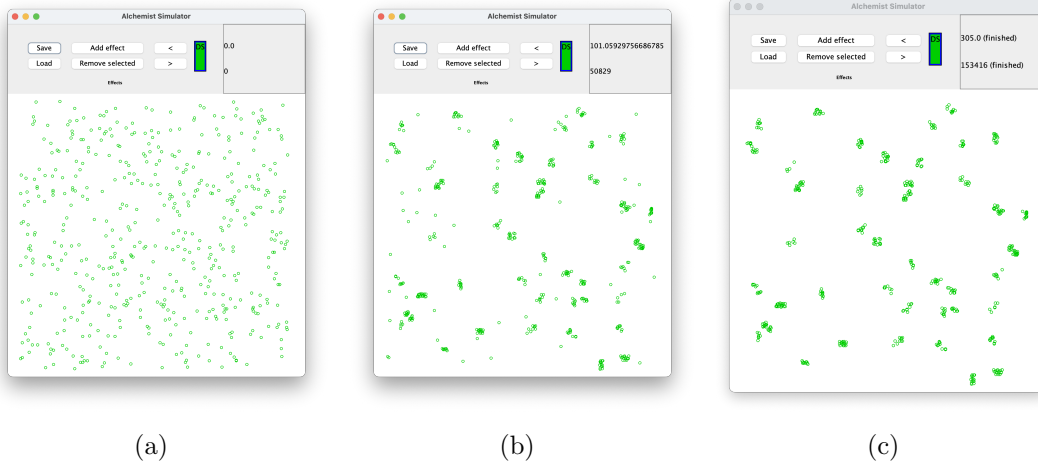


Figura 6.1: (a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi

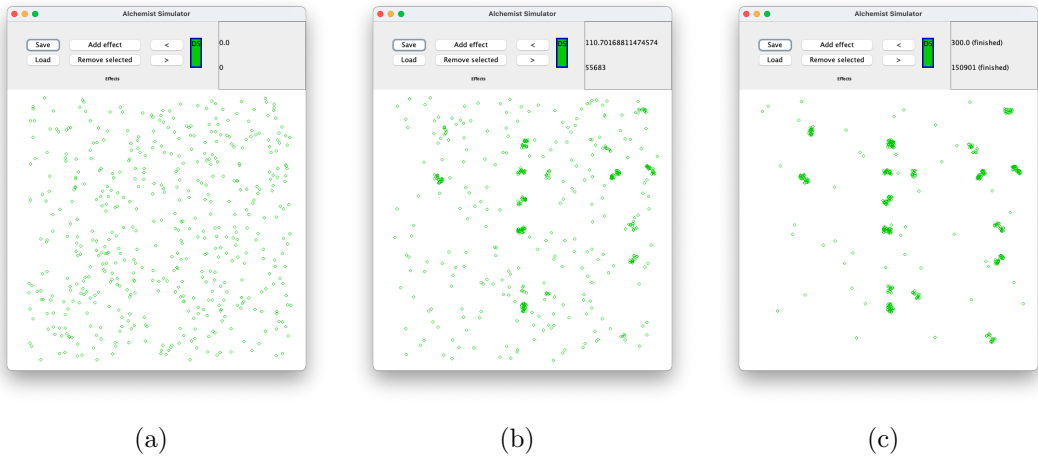


Figura 6.2: (a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi

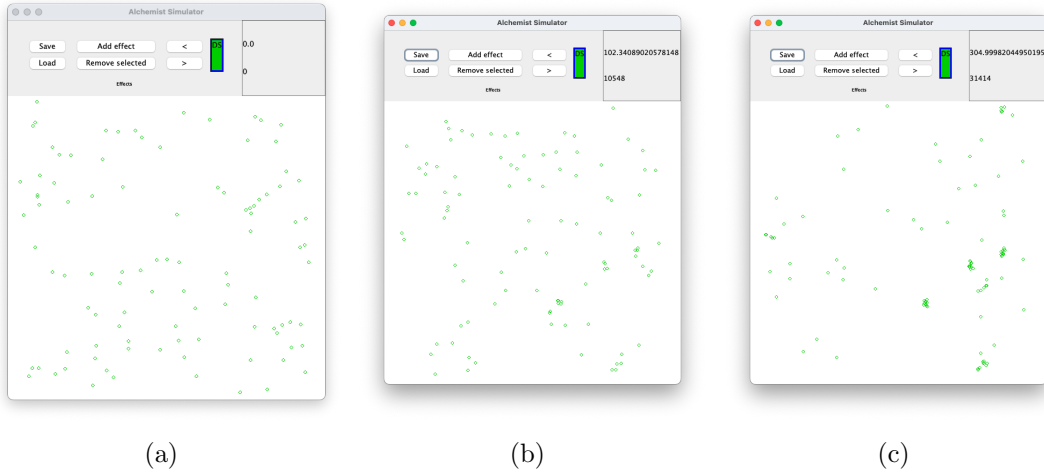


Figura 6.3: (a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi

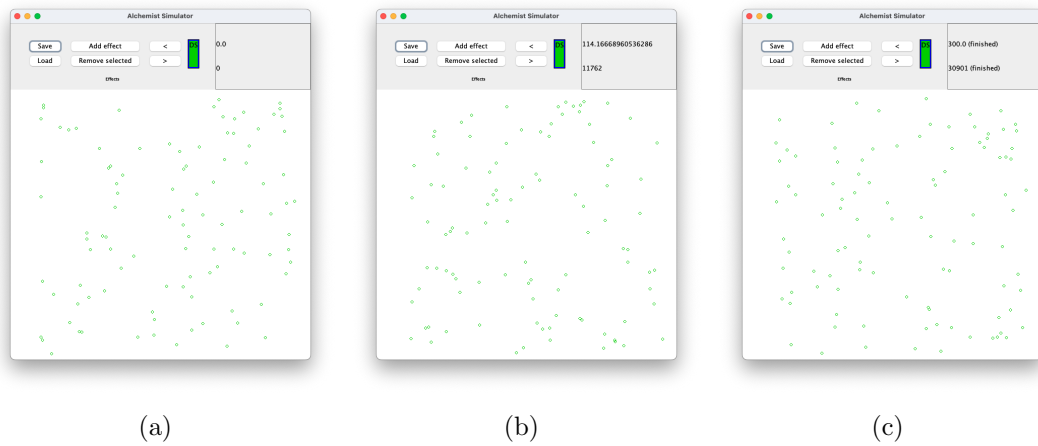


Figura 6.4: (a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi

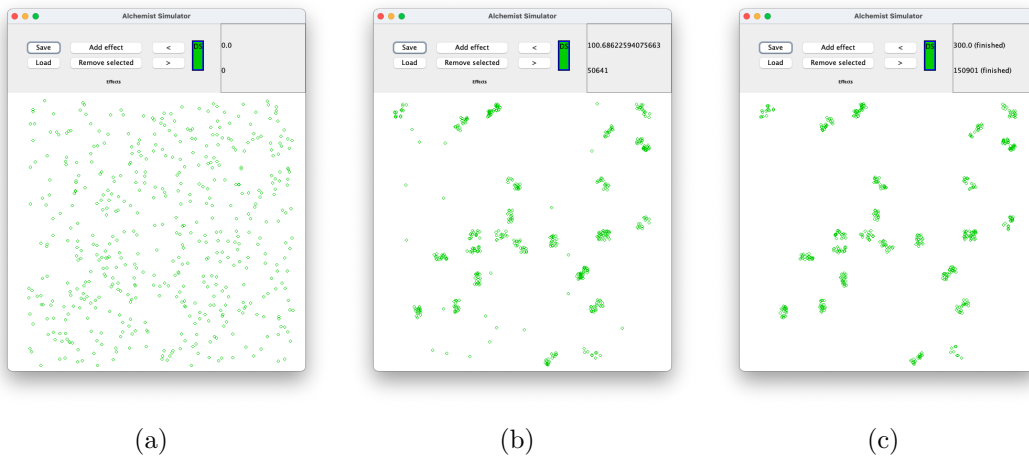


Figura 6.5: (a) Inizio (b) Dopo 100 secondi (c) Dopo 300 secondi

Capitolo 7

Conclusioni e lavori futuri

7.1 Lavori futuri

Un possibile sviluppo futuro di questo lavoro di tesi potrebbe consistere nel modificare il modello per permettere all'utente di definire l'angolo di scoperta del feromone. Attualmente ogni nodo “sniffa” il feromone a 360 gradi, ma sarebbe interessante poter definire un angolo di visuale, in modo da poter simulare comportamenti più realistici.

Capitolo 8

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.