ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea in Ingegneria e Scienze Informatiche

Simulazione event-driven di folle con micro-interazione fisica ed elementi cognitivi

Tesi di laurea in
Programmazione ad oggetti

Relatore
Prof. Danilo Pianini

Candidato
Kelvin Oluwada Milare
Obuneme Olaiya

I Sessione di Laurea
Anno Accademico 2021-2022

Sommario

La simulazione computerizzata è uno strumento per prevedere il comportamento di un sistema quando il sistema non è indagabile. Molti sono gli ambiti in cui viene utilizzata. Uno fra questi è l'evacuazione di folle. Le numerose tragedie, conseguenti a situazioni o eventi in presenza di folle, che hanno luogo in ogni parte del mondo a causa dell'incapacità di reagire in maniera coordinata a situazioni di emergenza, rendono necessario un innalzamento della comprensione delle dinamiche che emergono durante l'evacuazione. Molti studi hanno analizzato quelli che sono gli aspetti psicologici dell'essere umano che entrano in gioco in situazioni di panico. Ma è anche necessario considerare che cosa accade a un livello fisico. Questo contributo ha come obiettivo l'introduzione, all'interno del simulatore Alchemist, di elementi di micro-interazione fisica che possano fare da ponte a ciò che accade a livello cognitivo, così che sia possibile far emergere caratteristiche osservabili, come spinte e cadute, nell'ottica di aumentare il realismo nelle simulazioni di evacuazione di folle.

A chi ha creduto in me
e mi ha sempre sostenuto

Indice

Capitolo 1

Introduzione

1.1 Contesto e Motivazioni

Nello studiare il comportamento delle folle nelle grandi manifestazioni, si cerca di comprendere quali siano i fattori che hanno il maggior impatto sulla sicurezza in caso di panico dovuto a una situazione di pericolo [37]. Tra i fattori identificati vi sono la tipologia dell'evento, le strategie e le misure per il contenimento della folla, condizioni atmosferiche e le dimensioni della folla. Tuttavia è necessario approfondire ulteriormente quali siano le dinamiche che hanno luogo in queste situazioni. Ciò permetterebbe di stimare a priori l'entità di un eventuale disastro nonchè di prepararsi progettando gli spazi per il pubblico in modo da ridurre l'enità di un eventuale disastro e organizzando un servizio di pronto soccorso più rapido ed efficiente. La psicologia dell'essere umano è un aspetto determinante che, in situazioni di panico, se non ben gestita, può vanificare il tentativo di messa in sicurezza. Secondo Zeitz et al. [37] per cambiare il comportamento collettivo della folla, ad esempio da uno stato di tranquillità a uno turbolento, è sufficiente che vi sia un gruppo di persone che assuma un comportamento anomalo che attiri l'attenzione delle altre persone e le coinvolga. Un esempio molto comune è la rissa, che spesso porta anche le persone non direttamente coinvolte a parteciparvi.

1.1.1 La psicologia delle folle

Gustave Le Bon, antropologo, psicologo e sociologo francese, fu colui che per la prima volta parlò di *Mentalità di gruppo*. Nel suo libro *Psicologia delle folle* [21], Le Bon spiega che gli individui perdono il proprio senso di responsabilità quando si trovano a far parte di un grande raduno di persone. In queste situazioni emergono comportamenti primitivi agevolati dal sentimento d'invincibilità e dalla suscettibilità al contagio sociale.

Teoria della predisposizione La *teoria della predisposizione* [24] è un tentativo alternativo a quello di Le Bon di spiegare il comportamento delle folle. Secondo questa teoria l'azione collettiva è spiegata in termini di tendenze individuali preesistenti che indicano che i comportamenti violenti derivano da delle personalità cosiddette "anti-sociali" (non in linea con quelle della collettività).

Teoria della norma emergente Un ulteriore tentativo è la *Teoria della norma emergente* [22]. La teoria della norma emergente suppone che i comportamenti anomali osservabili nelle folle originino da nuove norme comportamentali emergenti, in risposta a una crisi improvvisa. La folla tende a concentrarsi maggiormente sui comportamenti anormali e a conformarsi a essi. Questo denota quindi una sorta di pressione sociale che porta alla conformità. Questo processo di conformazione alla nuova norma comportamentale è tanto più rapido quanto più sono agevolate le comunicazioni tra gli individui.

1.1.2 La simulazione e le sue applicazioni

A volte può accadere che per un problema si abbiano a disposizione più soluzioni. Individuare la migliore richiede che vengano definiti criteri di ottimalità, che vengano fatte delle stime sulla qualità delle possibili soluzioni che si hanno a disposizione e, infine, che si confrontino le soluzioni sulla base del risultato atteso [20]. Spesso, però, la natura del problema che si vuole risolvere, sia per questioni economiche che per questioni etiche, non permette di fare delle prove direttamente "sul campo". Per questo, risulta utile riprodurre il problema in un contesto controllato dove sperimentare varie soluzioni, analizzandone i costi e i benefici. Le simulazioni, sono utlizzate in molteplici ambiti applicativi: dai sistemi di produzione ai processi chimici, dalle reti di comunicazione ai trasporti. Anche lo studio del comportamento delle folle rientra tra le applicazioni della simulazione. Oltre a essere interessante, perché riguarda il nostro modo d'interagire con l'ambiente e le persone che ci circondano, quello del comportamento delle folle è un tema che è rilevante in più ambiti, che spaziano dalla sicurezza alla riproduzione accurata di scenari di vita quotidiana in sistemi videoludici.

1.1.3 Disastri agli eventi di massa

Numerosi sono i disastri che si sono verificati nel corso degli anni agli eventi di massa. Tra il 1971 e il 2011 è possibile identificarne 156 [30]. Spesso la gravità del disastro si amplifica per via di una perdita di razionalità delle persone [12]. Capita che un pericolo, percepito in una zona ristretta dell'ambiente, porti a una tragica isteria di massa. L'isteria porta le persone a spingersi tra loro nel tentativo di fuggire, facendole cadere a terra generando morti e feriti.

Figura 1.1: Disastro della *Love Parade* del 2010

Love Parade Il "Love Parade" è stato un popolare festival di musica elettronica e technoparade svoltosi ogni anno a Berlino a partire dal 1989 fino al 2010 (tranne che negli anni 2004, 2005, e 2009 i cui eventi sono stati cancellati). L'ultimo evento, svoltosi nel 2010 a Duisburg, fu teatro di una tragedia. L'area dedicata al festival era di circa 100,000 metri quadrati. Per l'evento erano previsti fino a 1,4 milioni di visitatori. Sebbene le autorità regolatorie abbiano espresso le loro perplessità sulla sicurezza dell'evento, la città di Duisburg approvò comunque l'evento limitando però il numero di persone a 250.000. L'unico punto d'ingresso e di uscita fu un tunnel lungo 240 metri con una rampa. Fin dalle prime ore l'afflusso di persone divenne sempre più sostenuto. La presenza nel tunnel di ostacoli e veicoli per il trasporto della strumentazione musicale creò una situazione di congestione. Nel tentativo di alleviare il sovraffollamento, la polizia all'ingresso iniziò a dare istruzioni ai nuovi arrivati tramite altoparlante per farli tornare indietro. La gente continuò a spingere nello spazio ristretto del tunnel. La dinamica ha portato al decesso di 21 persone e ad almeno 500 feriti. Sulla causa dei decessi alcuni suggerirono che fu per via delle cadute nel tentativo di fuga dal tunnel. Le autopsie hanno dimostrato che i decessi erano dovuti a gabbie toraciche schiacciate [15].

Piazza San Carlo In occasione della finale di Champions League, il 3 Giugno 2017 in piazza San Carlo a Torino venne allestito un maxi-schermo per seguire la

Figura 1.2: Disastro Piazza San Carlo

partita in diretta. Un gruppo di persone ha utilizzato lo spray urticante generando il panico tra la folla[1]. Gli oltre 30.000 presenti hanno cercato di fuggire in maniera disordinata. La fuga ha provocato la morte di due persone e oltre 1500 feriti.

1.1.4 La simulazione come strumento di prevenzione

Non è raro ritrovarsi in situazioni ricreative dove il numero di persone coinvolte è elevato. Basti pensare a manifestazioni sportive o di piazza, feste e sagre, concerti ma anche a luoghi apparentemente meno affollati come la scuola o il luogo di lavoro. Tutte queste situazioni hanno in comune il fatto che in caso di emergenza è necessario attuare un piano di evacuazione per la messa in sicurezza degli individui interessati. La riuscita o meno dell'evacuazione dipende in buona sostanza da come ci si è preparati ad affrontarla. Spesso, per testare l'efficacia delle misure di prevenzione e addestrare le persone a procedere in maniera ordinata in caso di emergenza, vengono attuate delle prove di evacuazione. Tuttavia il difetto di queste simulazioni è che nella maggioranza dei casi non catturano alcuni aspetti psicologici. Le condizioni psicologiche di un individuo cambiano a secon-

[1] https://web.archive.org/web/20220705095648/https://www.lastampa.it/torino/2022/01/21/news/definitive_le_condanne_per_la_banda_dello_spray_della_strage_di_piazza_san_carlo-2837671/

da che ci si trovi di fronte a un reale pericolo o meno. Come già accennato, per ragioni etiche non si può pensare d'innescare un vero incendio o di allagare un edificio. È per questo che una particolare attenzione viene posta nella simulazione computerizzata.

1.1.5 La simulazione computerizzata

Le simulazioni portate avanti da un calcolatore presentano una valida alternativa alle simulazioni reali se si considerano il tempo assoluto e i costi necessari a eseguirle. Esistono diversi modelli di simulatori che hanno però il comune obiettivo di riprodurre in maniera quanto più fedele possibile una situazione reale. I modelli per la riproduzione delle dinamiche di folla si possono classificare secondo i seguenti criteri [5]:

- **Modello del dominio**: mappa i concetti del problema in entità simulabili. Esistono diverse strategie di modellazione che è possibile utilizzare separatamente o in combinazione e comprendono *automi cellulari*, *gas reticolari*, *forze sociali*, *fluidodinamica*, *ad agenti*, *teoria dei giochi* ed *esperimenti con animali*.

- **Individualità**: se non vengono mostrate caratteristiche peculiari di un pedone come l'età e il sesso, allora si sta parlando di pedoni *omogenei*. In caso contrario questi vengono detti *eterogenei*.

- **Scala**: alcuni modelli possono considerare l'insieme dei pedoni come un tutt'uno, questi si dicono *macroscopici*, mentre altri riescono ad apprezzare le interazioni al livello del singolo pedone quindi detti *microscopici*.

- **Spazio e tempo**: le variabili che le descrivono possono essere discrete o continue.

- **Emergenzialità**: la simulazione può voler riprodurre situazioni di *normalità* oppure di *emergenza*. Il comportamento nei due casi differisce molto.

Segue una panoramica sui principali tipi di approcci utilizzati in letteratura [38]:

Automi cellulari Furono introdotti per la prima volta da Von Neumann [33]. Nelle simulazioni basate sugli automi cellulari vengono considerate un insieme di celle rigidamente collocate in una griglia. Le simulazioni procedono con tempo discretizzato e a ogni step il valore di ciascuna cella viene aggiornato sulla base del valore delle celle del vicinato in base a un insieme di regole. Il più noto automa cellulare e il "Gioco della vita" di John Conway [18]. L'obiettivo di questo

automa cellulare era di mostrare che da poche semplici regole si possono ottenere comportamenti simili alla vita. Nell'ultimo decennio, gli automi cellulari, sono stati utilizzati per descrivere le dinamiche dell'evacuazione delle folle. Questi modelli sono riusciti a dimostrare sia come elementi dell'ambiente circostante, come la larghezza delle uscite e la disposizione degli ostacoli, abbiano un impatto sulla mobilità della folla [36].

Gas reticolari I gas reticolari sono un particolare caso di automi cellulari. Furono introdotti nel 1980 da Fredkin and Toffoli [8]. Nei modelli con gas reticolari ogni pedoni viene visto come una particella attiva nella griglia. Questi modelli utilizzano la probabilità e la statistica per studiare il comportamento delle folle, in particolare il loro movimento in diverse tipologie di edifici [17, 31, 25, 11].

Fluidodinamica Henderson [16] ritiene che le folle di pedoni a livello macroscopico si comportano come i fluidi e i gas. Bradley [3] ha ipotizzato che le equazioni di Navier-Stokes, che governano il moto dei fluidi, possano essere applicate per descrivere il moto dei pedoni. I modelli fluidodinamici mostrano la variazione nel tempo della densità e della velocità utilizzando equazioni differenziali alle derivate parziali. Questi modelli riescono a riprodurre le condizioni di alta pressione che sono tra le cause di morte nei disastri agli eventi di massa.

Modelli basati sulle forze sociali Furono proposti da Helbing nel 1995 [14]. Questi modelli tentano di riprodurre comportamenti realistici basandosi sulla combinazione di diversi stimoli e condizioni: la volontà di un pedone di raggiungere un particolare obiettivo, di mantenere una certa distanza da altri pedoni od ostacoli fino all'attrazione che un pedone prova per un altro pedone, come un amico o un genitore.

Modelli ad agenti Quello ad agenti è un modello computazionale che modella l'individuo attraverso un agente virtuale. Ogni agente è governato da regole che disciplinano le interazioni con altri agenti e con l'ambiente circostante permettendo di far emergere comportamenti collettivi come ad esempio il panico [26]. I modelli ad agenti sono computazionalmente più costosi degli altri tipi di approcci. Tuttavia, riescono con semplicità a simulare pedoni eterogenei grazie al fatto che ogni agente può essere configurato per avere un comportamento diverso da gli altri [38].

Modelli ibridi Secondo Xiaoping [35], la tendenza è quella di costruire modelli combinando più approcci. Ad esempio, automi cellulari in combinazione con gas reticolari, oppure modelli ad agenti basati su automi cellulari, e così via. Questo permetterebbe di compensare la difficoltà nel simulare il comportamento dei pedoni

e le limitazioni delle risorse computazionali sfruttando i vantaggi che i diversi modelli offrono. Di particolare interesse per questo contributo sono stati i modelli ad agenti basati sulle forze sociali.

1.1.6 Comportamenti emergenti nell'evacuazione di folle

Lo studio del comportamento delle folle mediante la simulazione ha permesso d'individuare fenomeni tipici che emergono durante le evacuazioni delle folle. Segue un elenco di alcuni di questi:

- **Intermittenza**: Un'analisi del disastro in Mina/Mecca nel gennaio del 2006 ha portato a osservare che, con un aumento della densità delle persone, da un flusso costante le persone iniziarono a muoversi con intermittenza [13];
- **Turbolenza**: Sempre analizzando il disastro avvenuto alla Mecca, si è osservato che, da un movimento a intermittenza, con un ulteriore incremento della densità, il moto della folla diventa sempre più irregolare e turbolento. Le persone si muovono in maniera casuale in ogni direzione spingendo gli altri. Di conseguenza, molti iniziano a inciampare. Per schivare i pedoni caduti, la folla inizia a rallentare e si viene a creare una pressione di alta intensità. Questa pressione non può far altro che costringere la folla a calpestare le persone ormai cadute [13].
- Effetto **"veloce è lento"**: è un effetto che emerge soprattutto in corrispondenza delle uscite, che, se non adeguatamente larghe, costituiscono un collo di bottiglia. Le persone impazienti cercano di muoversi più velocemente. Ciò che si è osservato è che la velocità media della folla però diminuisce. Questo può risultare molto pericoloso, se la folla sta fuggendo da un pericolo, ad esempio il fuoco. [10]

1.1.7 Alchemist

Alchemist [27] è un progetto *open-source* nato all'interno dell'Università di Bologna. Si tratta di un simulatore a eventi discreti (Descrete Event Simulator, DES) [1, 7]. L'evouluzione delle simulazioni avviene secondo delle distribuzioni di probabilità. Ciò significa che Alchemist è un simulatore stocastico e che l'evoluzione del sistema non è deterministica. Ogni evento viene eseguito secondo la propria distribuzione temporale, facendo avanzare il tempo della simulazione di conseguenza. Come il nome suggerisce, Alchemist è in grado di simulare scenari appartenenti al mondo della chimica. Ma in realtà Alchemist vuole essere un simulatore *general-purpose*. La chiave del'estensibilità del simulatore si trova nell'interpretazione che viene data agli elementi costitutivi del suo metamodello.

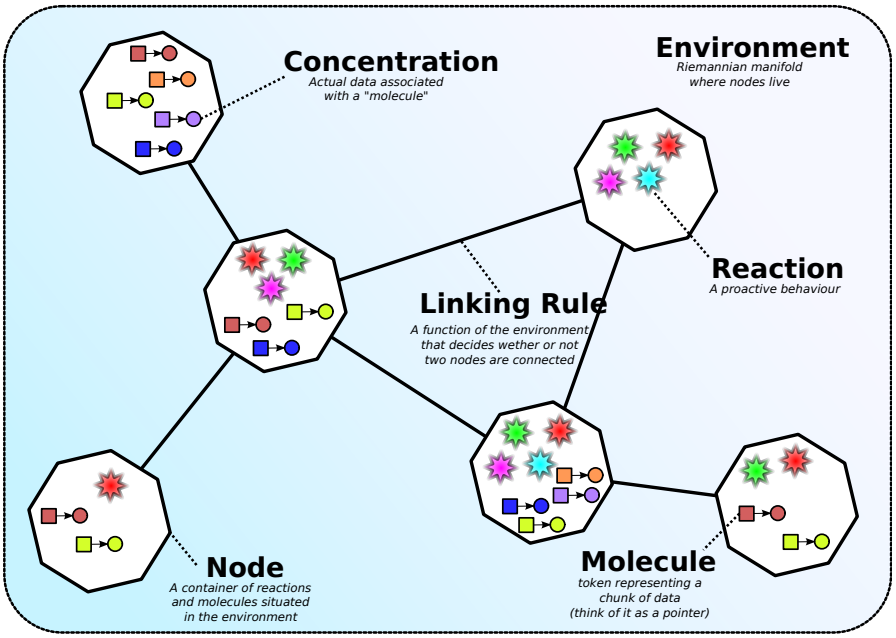

Figura 1.3: Metamodello di Alchemist

Tutto ciò che si può mappare nel metamodello di Alchemist è riproducibile dal simulatore, ivi incluso il movimento dei pedoni.

Il metamodello di Alchemist. Gli elementi costitutivi del metamodello di Alchemist sono:

- **Molecola**: è un etichetta. Corrisponde al concetto di variabile nei linguaggi di programmazione;
- **Concentrazione**: è il valore associato a una molecola. Nei linguaggi di programmazione corrisponde al valore di una variabile;
- **Nodo**: è un contenitore di molecole e reazioni posto all'interno di un ambiente;
- **Ambiente**: è un'astrazione dello spazio e un contenitore di nodi.
- **Reazione**: è un evento che può modificare l'ambiente. E costituita da azioni e condizioni. Ogni nodo contiene un insieme di reazioni che ne determinano il comportamento.
- **Condizione**: determina se una certa reazione dev'essere eseguita o meno.
- **Azione**: modella un qualsiasi cambiamento dell'ambiente.

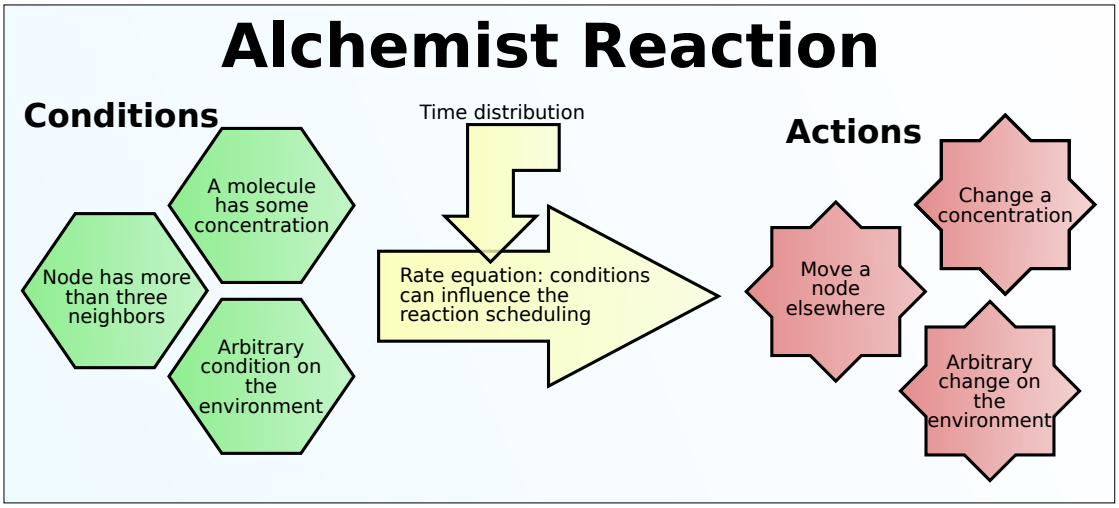

Figura 1.4: Schema sul funzionamento delle reazioni

Per maggiori informazioni consultare il sito ufficiale di Alchemist[2].

Le reazioni e il motore di simulazione Alchemist come simulatore DES ha a che fare con gli eventi. Se si guarda al meta-modello, gli eventi corrispondo alle Reazioni. Le reazioni comprendono:

- Un insieme di **Condizioni**: tramite un risultato booleano, queste determinano se la reazione, dato lo stato della simulazione, può essere eseguita o meno. L'insieme delle condizioni può essere anche vuoto. In tal caso, la reazione verrà eseguita secondo la sua distribuzione temporale.
- Un insieme di **Azioni**: a ogni azione corrisponde un cambiamento nell'ambiente di simulazione. Nel caso di simulazioni chimiche, può rappresentare un cambiamento della concentrazione di una molecola in un nodo. Nelle simulazioni dei pedoni invece può rappresentare uno spostamento del pedone.
- Una **Distribuzione temporale**: produce un *rate*, che formalmente, rappresenta il numero di volte per unità di tempo che la reazione deve essere eseguita, previa validità di tutte le sue condizioni.

Come già accennato, nella rappresentazione del tempo i simulatori si dividono in simulatori **continui** e **discreti**. Nei primi, il tempo è una variabile continua cosicché il sistema possa evolvere continuità. Nei secondi il tempo è discreto. Tra i

[2]https://alchemistsimulator.github.io/

simulatori discreti vi è un ulteriore divisione. Infatti, la discretizzazione del tempo può venire a priori scegliendo un intervallo di tempo Δt chiamato *tick* secondo il quale il sistema viene aggiornato. Qualora due reazioni debbano essere eseguite nello stesso tick, da fuori vengono viste come contemporanee. è importante scegliere con attenzione la durata del tick in quanto con un intervallo troppo tasso il sistema verrebbe aggiornato troppe volte con un conseguente abbassamento delle prestazioni. Al contrario con un intervallo troppo alto a risentirne sarà la precisione.

Alchemist non adotta questo approccio, chiamato *Time driven*, ma come già accennato utilizza un approccio a *Descrete Event Simulation*. Con questo approccio le reazioni (eventi) vengono eseguite una a una secondo l'ordine dei tempi di programmazione basato sull'instante in cui, in assenza di reazioni da eseguire, la distribuzione temporale indica che debba essere eseguita(τ). Una volta eseguita la reazione, il tempo della simulazione viene incrementato di un intervallo alla durata dell'esecuzione della reazione.

Alchemist utilizza il *Next Reaction Method* [9] di Gibson e Bruck, che a ogni passo della simulazione, con l'appoggio di una coda di priorità indicizzata e un grafo delle dipendenze, determina la prossima reazione da eseguire. La chiave di questo metodo è la velocità con cui vengono aggiornate le strutture interne che richiedono una complessità logaritmica.

Il grafo delle dipendenze è un grafo diretto dove i nodi sono le reazioni e gli archi connettono una reazione r a tutte quelle che dipendono da essa. Ogni volta che viene eseguita la reazione r le reazioni che dipendono da essa vengo aggiornate e opportunamente riprogrammate. Per gestire in maniera efficiente questo problema è stato introdotto il concetto di *contesto*. Il contesto può essere *locale*, *di vicinato* (neighborhood) o *globale*. Ogni reazione ha un contesto di *input* e uno di *output* che indicano rispettivamente da dove provengono le modifiche che influenzano il calcolo del nuovo τ e dove verranno effettuate le modifiche all'esecuzione della reazione.

1.1.8 Simulazioni di pedoni cognitivi in Alchemist

In Alchemist è presente un modulo per la simulazione degli agenti cognitivi [23]. Gli agenti vengono modellati secondo le caratteristiche presenti nel modello *IMPACT* [34] e comprendono: età, sesso, velocità, grado di conformità alle regole e l'attitudine ad aiutare. Vi sono altre caratteristiche che invece provengono dai principi del *Network Oriented Modelling* [32] e comprendono: sensazione di pericolo, livello di paura, desiderio di fuggire o non fuggire e intenzione di fuggire o non fuggire. Per comprendere la differenza tra il concetto di desiderio e quello d'intenzione si pensi ad esempio a un pedone bloccato a terra in seguito a una

caduta. Questo ha mentalmente un forte desiderio di fuggire, ma non riuscendoci fisicamente ha al contempo una bassa intenzione di andarsene.

Gli agenti cognitivi in Alchemist si muovono secondo i comportamenti di *steering* introdotti da C. W. Reynolds [28]. Questi hanno lo scopo di attribuire realismo ai movimenti degli agenti. Si basano su una semplice formula:

$$steering = desired\_velocity - current\_velocity \tag{1.1}$$

dove *current_velocity* è il vettore che rappresenta il movimento che l'agente sta seguendo, mentre *desired_velocity* è il movimento che l'agente vorrebbe seguire per arrivare il prima possibile nel punto desiderato. Tra i comportamenti implementati in Alchemist troviamo: **raggiungere** o **fuggire** da un determinato punto d'interesse, **arrivare** in un determinato punto d'interesse (si tratta di raggiungere il punto d'interesse, rallentando fino ad avere velocità zero in corrispondenza dell'obiettivo). **girovagare** ed **evitare gli ostacoli**.

Utilizzo di Alchemist. Per poter costruire delle simulazioni, Alchemist richiede la scrittura di un file di configurazione scritto in YAML[3]. YAML è un formato di serializzazione di dati facilmente leggibile dall'uomo. I file di configurazione sono delle mappe YAML le cui principali chiavi sono:

- **incarnation**: é l'unica chiave obbligatoria. Rappresenta un'istanza di metamodello di Alchemist che definisce il tipo di concentrazione.
- **seeds**: Alchemist è un simulatore stocastico, il che vuol dire il comportamento delle sue entità è spesso lasciato al caso. Tuttavia è talvolta necessario riprodurre determinate simulazioni. Per questo vengono impostati dei semi per la generazione di numeri casuali. Viene impostato un seme sia per la fase di creazione che per la fase di esecuzione della simulazione. Questi prendono il nome di *scenario* e *simulation*.
- **variables**: è una sezione che contiene valori che si intende riutilizzare in uno o più punti del file di configurazione.
- **environment**: l'ambiente dentro il quale si svolge la simulazione
- **deployments**: configurazione iniziale dei nodi, in particolare le loro posizioni, proprietà e comportamenti.

[3] `https://yaml.org/`

Capitolo 2

Micro-interazione fisica nella simulazione di folle

2.1 Analisi

L'idea del contributo di questa tesi è trovare il modo di arricchire lo strato cognitivo dei pedoni tenendo conto anche di alcune micro-interazioni fisiche. Per questo lo strato cognitivo e quello fisico vengono pensati separati. In particolare il primo attraverso regole psicologiche ed emotive, sceglie qual è il prossimo punto in cui desidera andare. Questo punto viene poi preso in considerazione e aggiustato da uno strato fisico, che tiene conto dei limiti fisici che il pedone potrebbe incontrare (collisioni con muri o altri pedoni).

Un modello di forze sociali generalizzato Helbing [10] ha proposto un modello rappresentativo del comportamento dello folle in situazioni di panico. Il modello è il risultato di un'analisi della letteratura in ambito socio-psicologico e dei materiali video disponibili che hanno permesso di formulare le seguenti assunzioni:

- Le persone si muovono o cercano di muoversi in modo considerevolmente più veloce rispetto alla norma
- La natura delle interazioni interpersonali è sempre più fisica e le persone iniziano a spingersi tra di loro.
- Il movimento e in particolare l'attraversamento di colli di bottiglia è scoordinato
- In corrispondenza delle uscite si osservano degli intasamenti e delle inarcature

- Le interazioni fisiche, sommate tra di loro possono generare una pericolosa pressione fino a $4,450Nm^{-1}$ [6] in grado di piegare le barriere d'acciaio e abbattere muri in mattone.
- L'evacuazione è rallentata per via dei feriti e delle persone cadute che agiscono da "ostacoli"
- Le persone tendono ad assumere il comportamento di massa, cioè "fare ciò che fanno gli altri"
- Le uscite alternative vengono di solito ignorate e poco utilizzate durante le situazioni di emergenza

Il modello che ne deriva considera che ogni pedone i di massa m_i:

- vuole muoversi con una certa *velocità desiderata* v_i^0 in una direzione desiderata e_i^0. Per questo cerca, in un tempo caratteristico τ_i, di adattare la sua velocità attuale v_i in modo da raggiungere quella desiderata;
- cerca di mantenere una distanza, dipendente dalla velocità, dai muri e da altri pedoni. Ciò darà luogo a due forze d'interazione rispettivamente f_{iW} e f_{ij}.

L'equazione dell'accelerazione che descrive come varia la velocità del pedone è dunque:

$$m_i \frac{dv_i}{dt} = m_i \frac{v_i^0(t)e_i^0(t) - v_i(t)}{\tau_i} + \sum_{j(\neq i)} f_{ij} + \sum_W f_{iW} \tag{2.1}$$

Le due forze aggiuntive f_{iW} e f_{ij} rappresentano rispettivamente la risposta alla compressione del corpo che avviene durante il contatto tra due pedoni o tra un pedone e un muro, e la frizione che cerca d'impedire che due corpi (muri compresi) a contatto scivolino tangenzialmente. Sono ispirate all'interazioni granulari [29, 4].

Modello fisico del movimento del pedone cognitivo Per il movimento si è preso spunto dal modello *HiDAC* [26] di Pelechano et al. Per modellare il movimento di un pedone i si considera la posizione che il pedone desidera raggiungere (F_i^{At}) in un certo step n. Viene poi tenuto conto di altri fattori psicologici e geometrici. Questi fattori includono la tendenza di mantenere una certa distanza da muri(F_{wi}^{Wa}), ostacoli(F_{ki}^{Ob}) e altri pedoni(F_{ji}^{Ot}).

$$F_i^{To}[n] = F_i^{To}[n-1] + F_i^{At}[n]w_i^{At} + \sum_w F_{wi}^{Wa}[n]w_i^{Wa} + \sum_k F_{ki}^{Ob}[n]w_i^{Ob} + \sum_{j(\neq i)} F_{ji}^{Ot}[n]w_i^{Ot} \tag{2.2}$$

In Alchemist, attualmente non vengono tenuti in considerazione gli ostacoli mobili, come invece accade in *HiDAC*. Inoltre tendenza di un pedone di mantenere un certa distanza dai muri viene modellata mediante uno dei comportamenti di steering introdotti in 1.1.8. La forza risultante influenzerà il risultato di F_i^{At}. Dunque nel nostro caso l'equazione che governa il movimento di un pedone risulta:

$$F_i^{To}[n] = F_i^{To}[n-1] + F_i^{At}[n]w_i^{At} + \sum_{j(\neq i)} F_{ji}^{Ot}[n]w_i^{Ot} \tag{2.3}$$

il versore della forza è quindi:

$$f_i^{To} = \frac{F_i^{To}}{|F_i^{To}|} \tag{2.4}$$

Così che la nuova posizione del pedone i può essere calcolata come segue:

$$p_i[n+1] = p_i[n] + \alpha_i[n]v_i[n]((1-\beta_i[n])f_i^{To}[n] + \beta_i[n]F_i^{Fa}[n])T + r_i[n] \tag{2.5}$$

dove:

- $v_i[n]$ e il modulo della velocità allo step n della simulazione;
- r_i e il risultato della forza di repulsione a cui e soggetto un pedone quando si sovrappone con un altro pedone;
- α indica se il pedone allo step n si muoverà nella posizione desiderata oppure se sarà soggetto a una forza di repulsione:
  $\alpha_i = \begin{cases} 0 & |r_i| > 0 \\ 1 & altrimenti \end{cases}$
- β_i serve per dar priorità all'aggiramento dei pedoni che son caduti:
  $\beta_i = \begin{cases} 0.5 & distanza\ dal\ pedone\ caduto < 2m \\ 0 & altrimenti \end{cases}$

Forze di evitamento A ogni pedone i viene assegnata un'area d'influenza, in particolare un rettangolo. Qualora un altro pedone j entri nell'area d'influenza, entra in gioco la forza di evitamento. Per il calcolo di questa forza si tiene in considerazione la distanza tra il pedone i e il pedone j e l'angolo che le traiettorie dei due pedoni formano. In questo modo è possibile determinare quanta importanza dare all'evitamento del pedone. La forza di evitamento è una forza tangenziale il cui scopo è di curvare la traiettoria del pedone per evitare la collisione con i pedoni nell'area d'influenza. L'angolo compreso tra i vettori velocità dei due pedoni v_i e v_j indica se i loro movimenti sono confluenti oppure opposti. Questo angolo viene anche utilizzato per simulare il processo di decisione umana. Infatti, la maggior

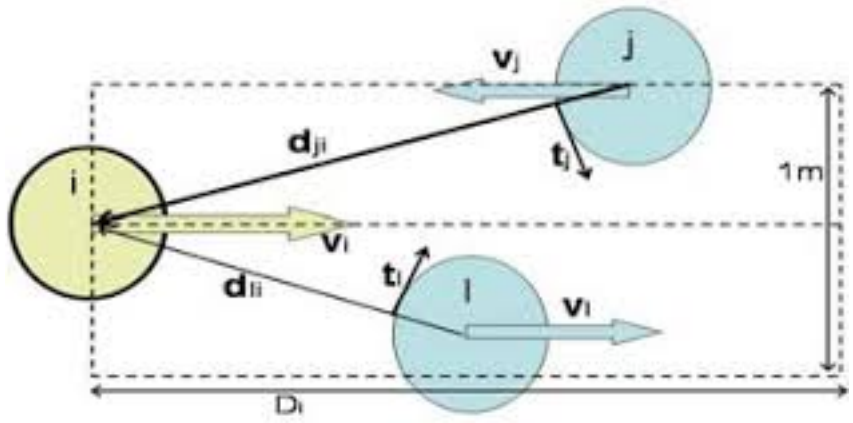

Figura 2.1: Area d'influenza del pedone

parte delle persone (occidentali) ha la tendenza a prendere la destra quando va in contro a un'altra persona. La forza tangente per l'evitamento è dunque:

$$t_j = \frac{(d_{ij} \times v_i) \times d_{ij}}{|(d_{ij} \times v_i) \times d_{ij}|} \tag{2.6}$$

Il vettore normalizzato verrà poi moltiplicato per due pesi per ottenere la forza di evitamento finale:

$$F_{ji}^{Ot} = t_j w_i^d w_i^0 \tag{2.7}$$

dove w_i^d è il peso relativo alla distanza tra i due pedoni e aumenta alla diminuzione della distanza tra i due pedoni.

$$w_i^d = (d_{ij} - D_i)^2 \tag{2.8}$$

e w_i è il peso relativo alla differenza in orientazione dei vettori velocità dei due pedoni. Distingue se i due pedoni si stanno muovendo nella stessa direzione oppure in direzione opposta. Perciò il modulo della forza in quest'ultimo caso sarà maggiore per evitare controcorrenti.

$$w_i = \begin{cases} 1.2 & (v_i \cdot v_j) > 0 \\ 2.4 & altrimenti \end{cases} \tag{2.9}$$

Infine D_i è il peso relativo alla percezione della densità della folla in un dato istante. Quando la folla è dispersa il pedone cercherà di evitare gli altri pedoni da una distanza maggiore rispetto a quando invece la folla è più fitta.

Forze di repulsione Quando un pedone si sovrappone con un ostacolo, un muro o un altro pedone, in *HiDAC* viene applicata una forza di repulsione. Per Alchemist, siamo interessati a introdurre la repulsione tra agenti, in quanto attualmente in simulatore non tiene in considerazione ostacoli in movimento e per i muri è già presente la gestione della collisione. Dunque nel nostro caso la forza di repulsione risultante sarà:

$$r_i[n] = \sum_{j(\neq i)} F_{ji}^{R_{Ot}}[n] \tag{2.10}$$

dove $F_{ji}^{R\_Ot}$ viene calcolata come:

$$F_{ji}^{R_{Ot}}[n] = \frac{(p_i[n] - p_j[n](r_i + \epsilon_i + r_j + d_{ji}[n])}{d_{ji}} \tag{2.11}$$

dove p_i è la posizione del pedone i e p_j la posizione del pedone j. I raggi r_i e r_j sono raggi dei pedoni i e j. d_{ji} è la distanza dal centro del pedone i a quello del pedone j. Infine ϵ_i, ϵ_j sono delle soglie di spazio interpersonali. Vengono utilizzati per differenziare i comportamenti di spinta dei diversi pedoni.

Spinte e cadute I pedoni possono assumere diversi comportamenti che possono essere innescati in qualsiasi momento. In situazioni controllate, i pedoni prima di muoversi aspettano che si liberi lo spazio per compiere il movimento, al contrario in situazioni di panico tenteranno di muoversi fino a che non collidono con altri pedoni che gli impediscano di proseguire. Combinando questi due comportamenti simultaneamente si osservano comportamenti emergenti dove alcuni pedoni che non rispettano lo spazio personale si avvicineranno ad altri pedoni spingendoli via per farsi strada nella folla congestionata. L'effetto di spinta viene ottenuto applicando la forza di repulsione e assegnando ai diversi pedoni soglie di spazio interpersonale differenti. Un pedone è sottoposto a una forza di repulsione quando il suo spazio personale viene occupato. Quindi un pedone i può spingere un altro pedone j senza anch'esso venga spinto e continuare per la sua traiettoria desiderata.

Nodo. Come già illustrato precedentemente, in Alchemist, un nodo è un contenitore di reazioni e molecole. Viene inserito in un ambiente e tramite quest'ultimo è possibile ricavarne la sua posizione nell'ambiente. Le reazioni, tramite una sequenza di azioni, sono in grado di modificare lo stato del nodo modificandone la posizione o impostandone alcuni parametri interni.

Tipologia di nodo. Per ogni simulazione sono disponibili uno o più tipi di nodi, a seconda dell'incarnazione. Ogni tipologia ne denota il tipico comportamento ed è orientato a un particolare tipo di simulazione. Nell'incarnazione *protelis* tra i tipi di

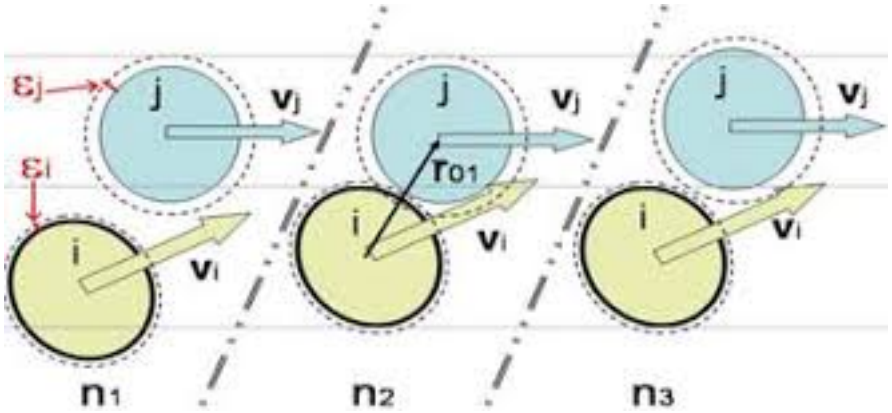

Figura 2.2: Soglia di spazio interpersonale

nodo disponibili troviamo il tipo *Pedestrian*, utilizzato nelle simulazioni di pedoni dotato di *attributi* che ad esempio ne descrivono la velocità di corsa e camminata. Un altro esempio è il tipo *Cell*, disponibile nell'incarnazione *biochemistry*, che modella lo stato di una cellula. Alcune reazioni e azioni, possono voler gestire sono un insieme ristretto di nodi. Una reazione che aggiunge un legame chimico in una cellula naturalmente non accetterà nodi di tipo *Pedestrian*.

2.2 Progettazione

Refactoring dell'architettura dei nodi Prima di procedere con l'implementazione degli elementi di micro-interazione fisica è necessario intervenire sull'architettura attuale dei nodi (fig. 2.3). Fin'ora, per garantire il riuso del codice, ogni volta che è stato introdotto un nuovo tipo di nodo si è fatto uso dell'ereditarietà. L'ereditarietà è un meccanismo offerto dal paradigma di programmazione ad oggetti che permette di estendere le funzionalità di una classe **base** attraverso un'altra classe. La classe che estende potrà contenere dei metodi aggiuntivi oppure, più semplicemente, delle ridefinizioni dei metodi della classe base. Queste ridefinizioni prendono il nome di **override**. Il vantaggio di questo meccanismo consiste nel riutilizzo del codice della classe base. Tuttavia in una gerarchia estesa come quella dei nodi di Alchemist questo meccanismo può compromettere la manutenibilità del codice. Il problema risulta più evidente nei linguaggi di programmazione che non godono di ereditarietà multipla come quelli con cui Alchemist è stato sviluppato. Osservando l'attuale gerarchia dei nodi è possibile notare l'esplosione combinatoria dei tipi di nodi. In generale, dati dei nodi di tipo A e di tipo B può essere utile introdurre un nodo di tipo AB. Analogamente dato dei nodi di tipo A, B e C, oltre al nodo di tipo AB potrebbe essere necessario introdurre i tipi AC, BC

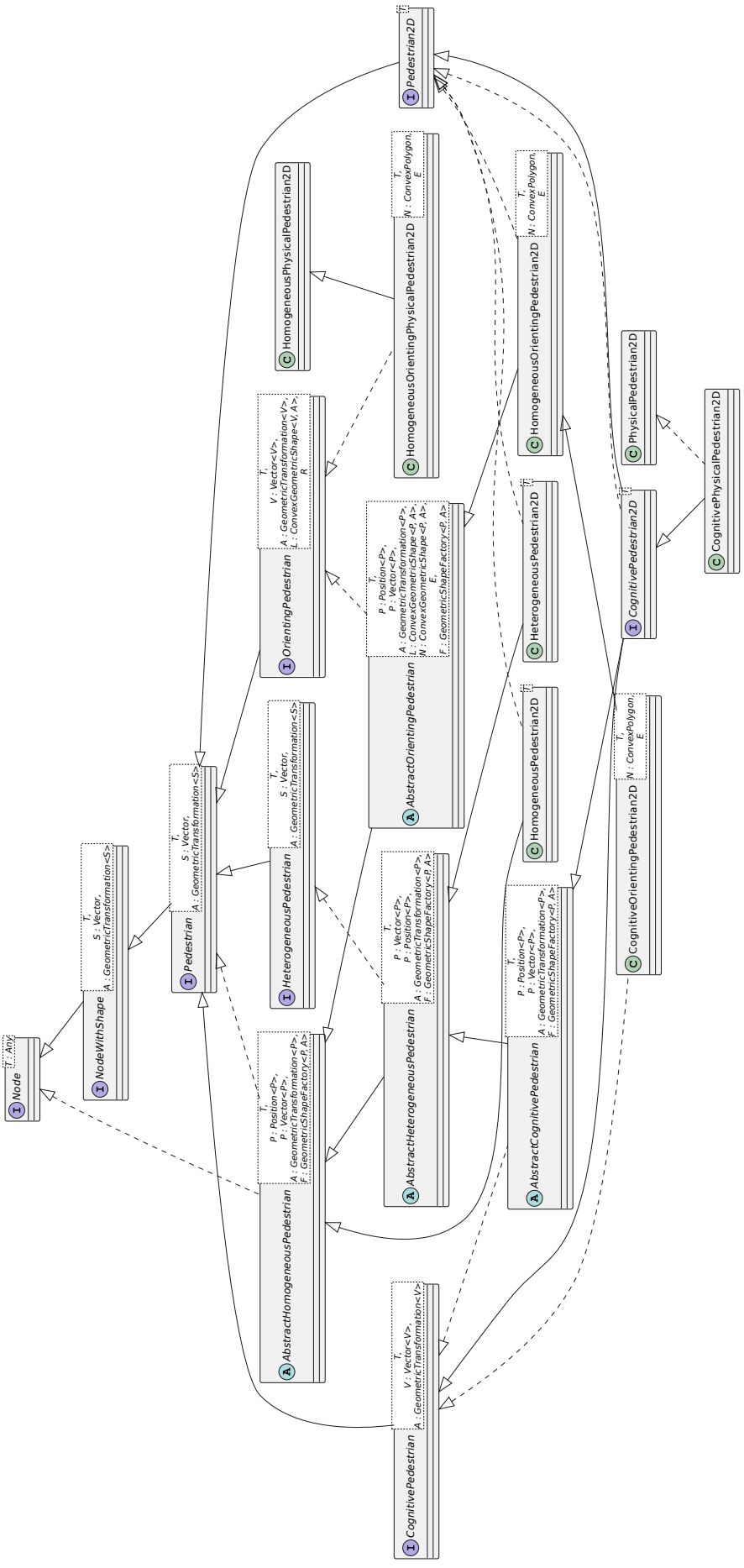

Figura 2.3: Gerarchia dei nodi pedoni in Alchemist precedente al refactoring

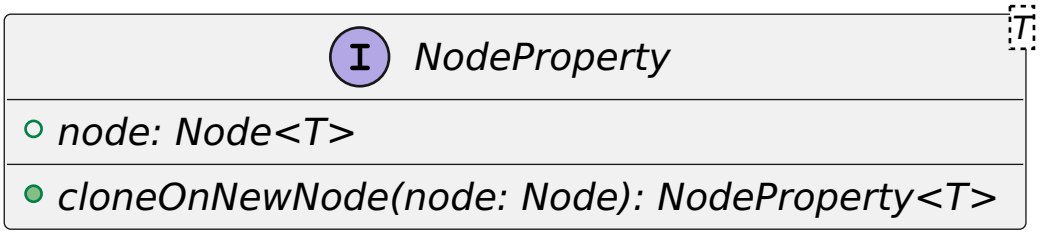

Figura 2.4: Interfaccia NodeProperty

e ABC. E così via. Non è sempre questo il caso, ma con l'aumentare del numero di nodi di base nel caso peggiore si avrebbe a che fare con 2^n tipi di nodo. Altra problematica può sorgere quando è necessario modificare l'implementazione della classe base. Ciò porterebbe a un intervento a cascata sulle classi figlie che con la base hanno un *dipendenza*.

Una soluzione per questi problemi può essere quella di utilizzare la composizione invece che l'estensione [2]. L'uso della composizione prevede che venga utilizzata l'ereditarietà solo a livello d'interfaccia. La classe che estende manterrà un riferimento interno alla classe base. Il riuso delle implementazioni avverrà quindi per delegazione.

Proprietà di un nodo. Osservando l'attuale gerarchia dei nodi è possibile dedurre che ogni tipo di nodo descrive quali sono le proprietà del nodo stesso. Ad esempio, il tipo *Cell*, come già detto, descrive le proprietà cellulari del nodo, il tipo *Pedestrian* racchiude le caratteristiche principali di un pedone e così via. Quindi, una possibilità di semplificazione dell'architettura su questi nodi può nascere utilizzando un approccio di configurazione mediante la composizione. Queste proprietà vengono rappresentate per mezzo dell'interfaccia *NodeProperty*. Figura 2.4

Nodi configurabili L'introduzione delle *NodeProperty* ha permesso di basarsi su una semplice implementazione di un tipo nodo generico, che potrà essere configurato aggiungendo a quest'ultimo le proprietà d'interesse. L'interfaccia *Node* è stata adattata per permettere la configurazione del nodo tramite l'aggiunta di *NodeProperty*. Infine sono stati aggiunti i metodi per ottenerle tutte o una in particolare. Da notare come non sia presente un metodo per la rimozione delle proprietà. La scelta è stata presa per evitare che si vengano a creare situazioni inconsistenti nel corso della simulazione. Figura 2.7

Proprietà fisiche di un nodo L'interfaccia *PhysicalProperty* è stata introdotta per poter attribuire proprietà fisiche ad un generico nodo. Attualmente espone un

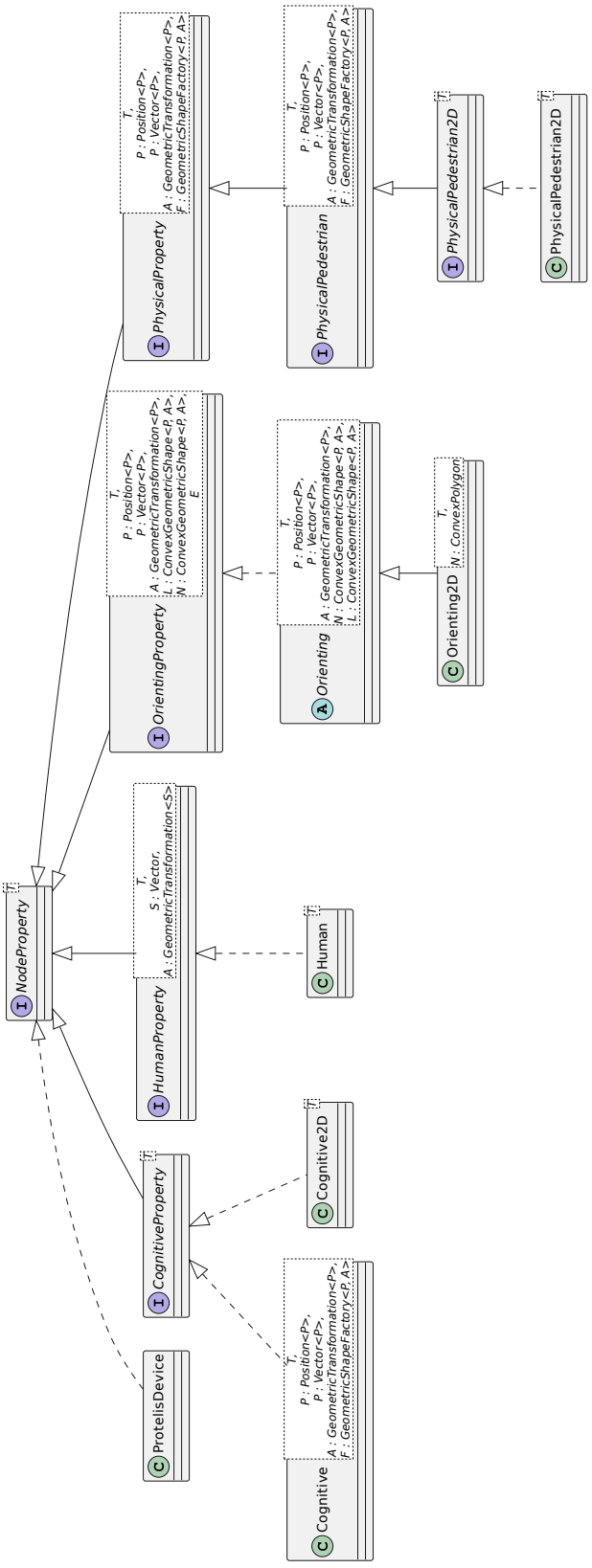

Figura 2.5: Implementazioni delle proprietà - parte 1

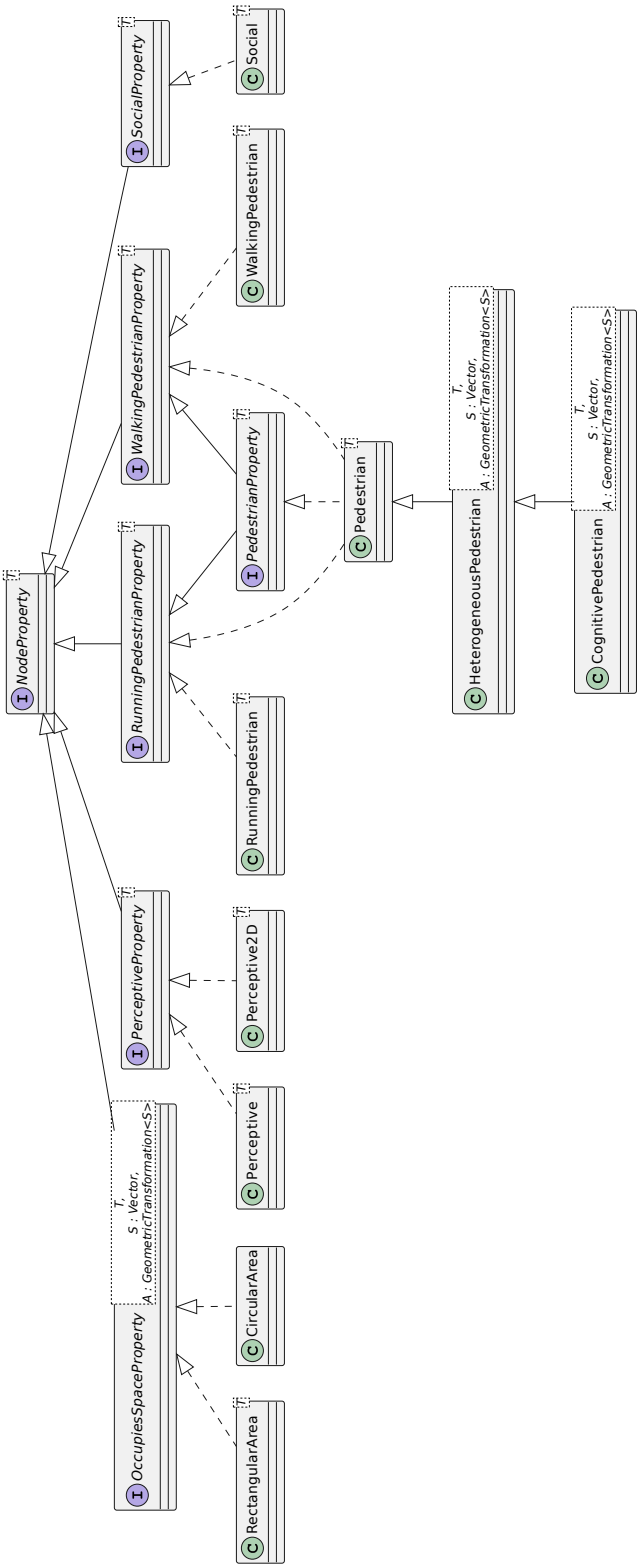

Figura 2.6: Implementazioni delle proprietà - parte 2

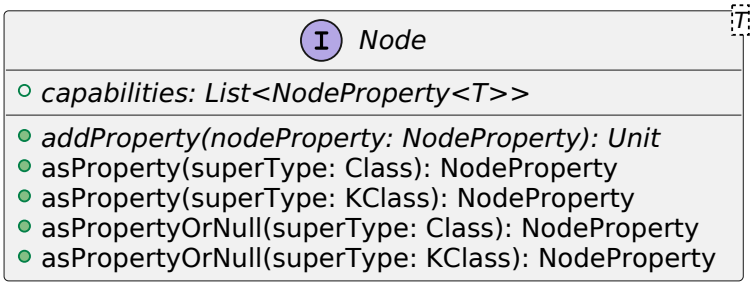

Figura 2.7: Nuovi metodi aggiunti all'interfaccia Node

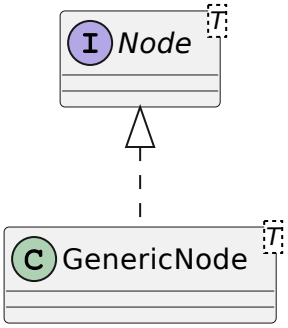

Figura 2.8: Gerarchia dei nodi successiva al refactoring

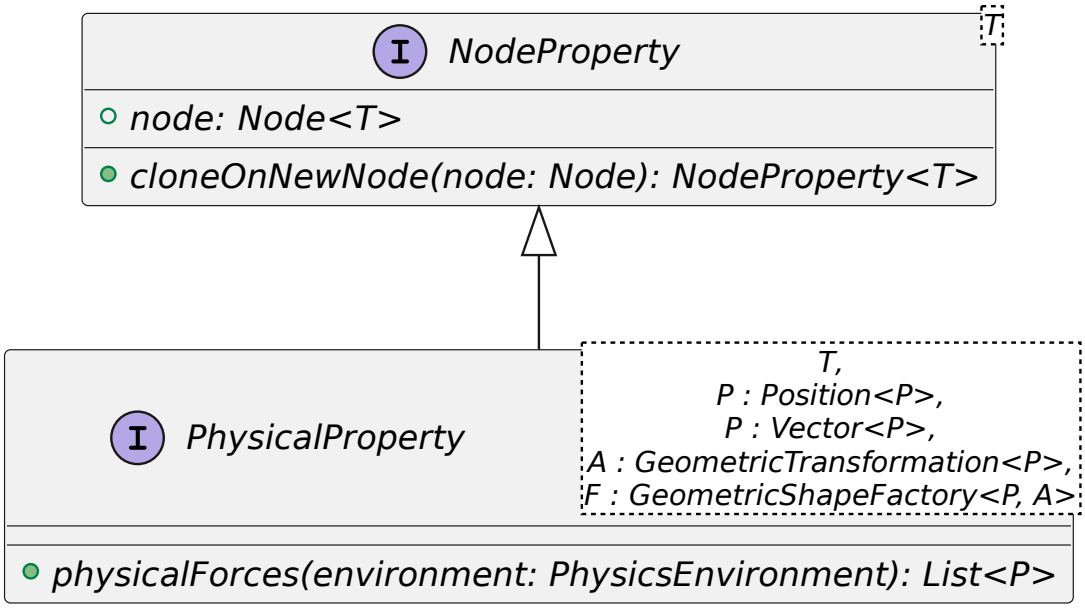

Figura 2.9: Interfaccia PhysicalProperty

metodo pubblico *physicalForces* tramite il quale è possibile ottenere una lista di forze vettoriali a cui il nodo è soggetto.

Pedone fisico *PhysicalPedestrian* estende l'interfaccia *NodeProperty*. Tra i suoi attributi troviamo:

- **comfortRay**: il raggio di comfort, ovvero la distanza che il pedone desidererebbe mantenere da altri ostacoli o pedoni.
- **comfortArea**: l'area di comfort che viene calcolata dato il raggio di comfort tenendo conto dello spazio occupato dal pedone stesso.
- **rectangleOfInfluence**: è il rettangolo dentro il quale gli altri pedoni esercitano un influenza sul pedone. In particolare il pedone cercherà di evitare e/o spingere i pedoni che entreranno in quest'area.
- **isFallen**: un attributo booleano che indica se il pedone risulta essere caduto.

I metodi *repulsionForces*, *avoidanceForce* e *fallenAgentAvoidanceForces* utilizzano il pattern *strategy* per calcolare la lista delle rispettive forze di repulsione ed evitamento. I metodi strategia sono *avoid* e *repulse* che dato un altro pedone calcolano modulo, direzione e verso delle forze. Un metodo privato prende in ingresso un

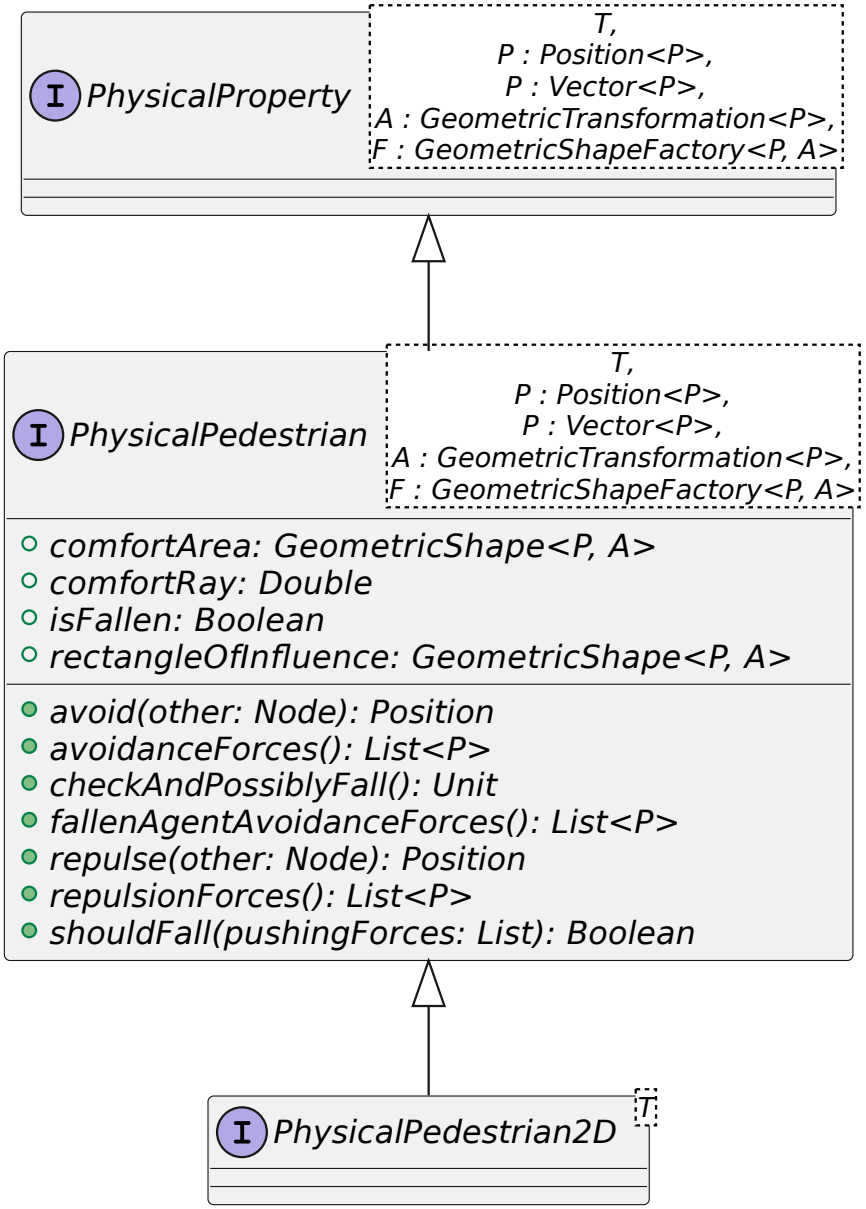

Figura 2.10: Interfaccia PhysicalPedestrian

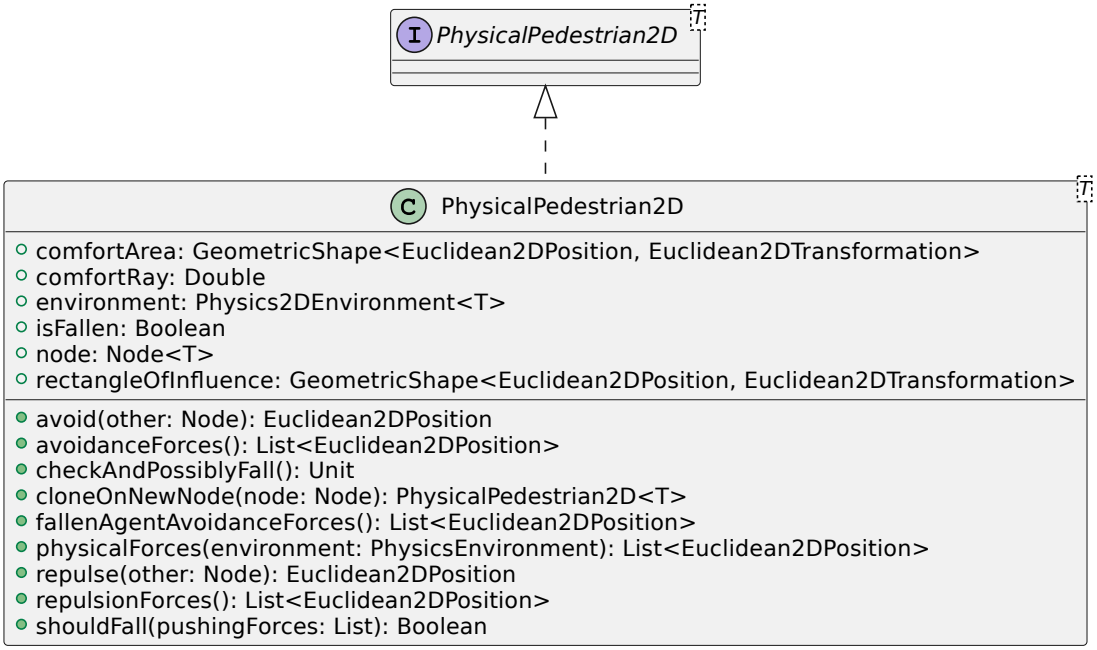

Figura 2.11: Implementazione 2D dell'interfaccia PhysicalPedestrian

area e una strategia. Vengono poi individuati i nodi presenti nell'area e restituito i risultati della strategia, raccolti in una lista.

Il metodo *checkAndPossiblyFall* stabilisce, qualora non fosse già caduto, se il pedone è nelle condizioni di dover cadere. Il metodo calcola la risultante delle forze di repulsione è se il modulo della forza risultante supera una certa soglia allora modifica lo stato del pedone. Come soglia è stata fissata una forza tale per cui il pedone soggetto a tale forza si sposterebbe con una velocità maggiore della sua caratteristica velocità in corsa.

Ambiente fisico Il compito d'individuare e risolvere le collisioni tra nodi è stato affidato a un ambiente fisico. Questo aggiornerà le posizioni e i vettori di velocità ogni qual volta si verifica una collisione. L'interfaccia *Dynamics2DEnviornment* definisce i metodi per settare e ottenere i vettori velocità dei diversi nodi. Definisce inoltre un metodo per aggiornare l'ambiente fisico (le posizioni) dato un intervallo di tempo trascorso. fig. 2.12.

Reazione per l'aggiornamento fisico nodo Per far si che il livello cognitivo del nodo comunichi allo strato fisico quale sia il punto in cui desidera andare è stata introdotta la reazione *PhysicalBlendedSteering*. Come suggerisce il nome, applica

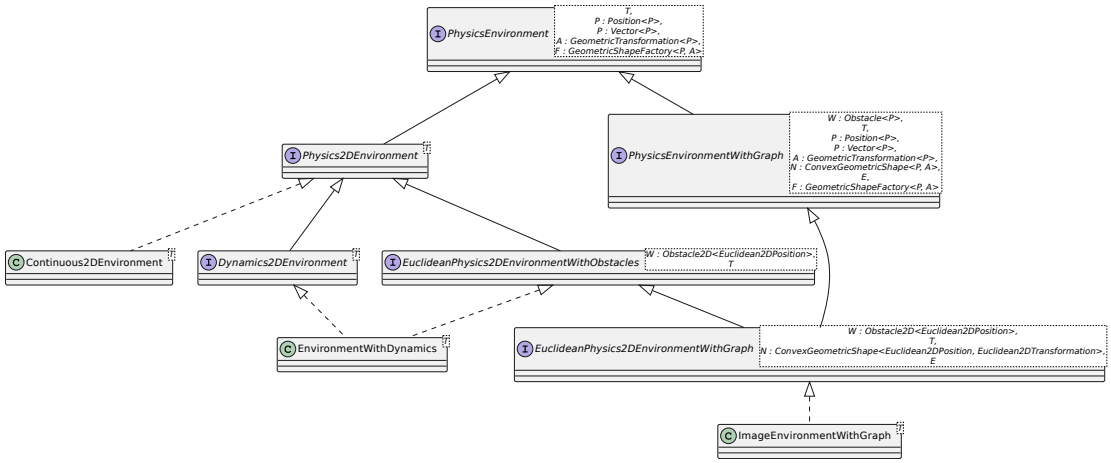

Figura 2.12: Il nuovo ambiente fisico

una strategia mista, in cui viene dato maggior peso alle forze che avvicinano il pedone al punto desiderato. La forza risultate viene poi utilizzata per il calcolo della nuova posizione secondo l'equazioni analizzate nella sezione precedente.

Reazione globale per l'aggiornamento dell'ambiente fisico In Alchemist, in genere una reazione appartiene a uno specifico nodo e aggiorna lo stato del nodo nell'ambiente. Per questioni di precisione, l'ambiente fisico prima di aggiornare lo stato interno (posizione dei nodi, rilevamento delle collisioni) dovrebbe aspettare che tutti i nodi abbiano aggiornato le loro proprietà fisiche. Per questo si è reso necessario introdurre il concetto di *reazione globale*. Una reazione globale ha dunque contesto globale. Ciò significa che la sua esecuzione potenzialmente comporterà l'aggiornamento di tutte le altre reazioni nella simulazione e che l'esecuzione delle altre reazioni provocherà l'aggiornamento della reazione globale. Per dare un ordine agli eventi schedulati è stato necessario specificare che tipo di dipendenza lega le reazioni locali al nodo alla reazione globale. Trattandosi di un aggiornamento della fisica dell'ambiente è stato dunque introdotto un nuovo tipo di dipendenza *PhysicsDependency*. La reazione globale dichiarerà di avere una dipendenza in uscita su *PhysicsDependency* mentre le reazioni locali al nodo la dichiareranno in entrata.

È stata introdotta l'interfaccia *Actionable*, che contiene i metodi comuni a *Reaction* e *GlobalReaction*. Queste estendono da *Actionable* aggiungendovi i metodi più specifici. Figura 2.13.

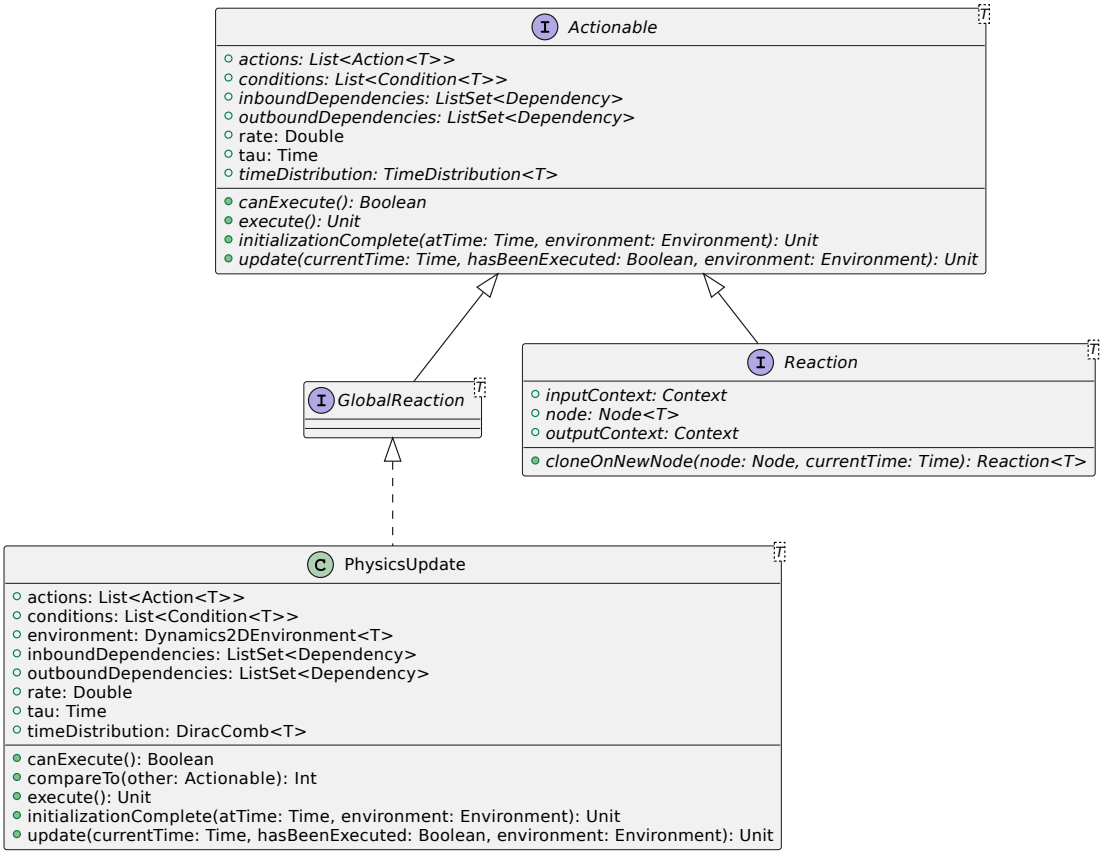

Figura 2.13: Gerarchia dell'interfaccia *Actionable*

2.3 Implementazione

2.3.1 Strumenti di sviluppo

Gradle Gradle è un tool open-source che permette di automatizzare il processo di build di un software. Come *make* per la compilazione di progetti scritti in linguaggio C/C++, permette di dichiarare quali sono le dipendenze (risorse esterne come ad esempio librerie) di cui il nostro software ha bisogno e si preoccupa di scaricarle qualora ce ne sia bisogno. Ma Gradle non si limita soltanto a gestire le dipendenze. In Alchemist, Gradle è configurato per, compilare il progetto, eseguire il controllo sulla qualità del codice, eseguire i test, generare la documentazione e il sito ufficiale, effettuare il rilascio degli artefatti su sistemi di distribuzione centralizzati e altro.

Kotlin Kotlin [19] è un linguaggio di programmazione multi-piattaforma, sviluppato dall'azienda JetBrains. Viene utilizzato sia come linguaggio orientato agli oggetti che come linguaggio funzionale. È un linguaggio open-source. Si appoggia alla *Java Virtual Machine* (JVM) e questo gli consente di essere interoperabile con Java. Sfrutta le numerose librerie presenti per Java e le estende. È un linguaggio espressivo e conciso che permette di ridurre la quantità di codice necessaria a svolgere un determinato task rispetto a Java. L'idea di Kotlin è di racchiudere in un unico linguaggio le migliori caratteristiche presenti negli altri linguaggi come *C#*, *Javascript* e *Python*.

Funzioni e proprietà di estensione Questo meccanismo trova la sua utilità quando si ha a che fare con librerie di terze parti, di cui non si ha controllo sull'API. Se per qualche motivo alcune classi della libreria mancano di alcune funzionalità, in Java ad esempio, due possibili soluzioni potrebbero essere quella di riscrivere la classe aggiungendo ciò che serve oppure estendere la classe con i meccanismi di ereditarietà. Queste due soluzioni presentano alcuni svantaggi: da una parte si viene meno al principio DRY (Dont Repeat Yourself) dato che si viene a creare una ripetizione del codice (senza considerare che se la classe è corposa, ciò può rappresentare un consistente dispendio di tempo); dall'altra abbiamo già visto che l'estensione è un meccanismo che va utilizzato il meno possibile per non compromettere la manutenibilità del codice.

Kotlin ha pensato a un modo per ovviare a questi due problemi: le funzioni/proprietà di estensione. Sono funzioni o proprietà che possono essere invocati come membri di una classe, ma che in realtà vengono definiti al di fuori di essa. Non vi è quindi duplicazione del codice o utilizzo di ereditarietà.

Delegazione Per ovviare alla fragilità causata dall'utilizzo dell'ereditarietà, Kotlin adotta alcune assunzioni di design. Ogni classe di default è considerata *final*. Ciò significa che non è possibile estenderla tramite ereditarietà. Se il progettista, scrive la classe in modo che sia predisposta per l'estensione, dovrà coscientemente aggiungere la keyword *open*. Esiste un pattern di design chiamato *Decorator*, utile per estendere le funzionalità di una classe, che non è stata pensata per essere estesa. Viene creata una nuova classe che implementa l'interfaccia della classe originale e che mantiene un'istanza della classe originale come membro interno, delegando a questa istanza i metodi che non sono stati alterati. In un linguaggio come Java, questo pattern richiede un certo grado di verbosità. Per questo Kotlin ha introdotto un supporto di prim'ordine per la delegazione. Nel dichiarare una classe, si indica che l'implementazione di una determinata interfaccia viene delegata a un altro oggetto. La classe quindi dovrà includere eventuali ridefinizioni dei metodi che vuole modificare. Sarà il compilatore a generare il codice necessario a chiamare i metodi dell'oggetto delegato.

Kotlin2PlantUML Alchemist è un progetto risalente al 2010. Nel tempo numerosi sono stati i contributi che lo hanno esteso. Spesso in progetti di grandi dimensioni è utile avere una fotografia dell'attuale architettura del software. Per questo corrono in aiuto i diagrammi UML (Unified Modelling Language). Costruire un diagramma delle classi manualmente, a partire dal codice sorgente di Alchemist, potrebbe richiedere molto tempo. Per questo è stato pensato di costruire una semplice libreria che genera in automatico file PlantUML[1]. Il progetto è open-source[2] e pubblicato in MavenCentral[3].

Git È un sistema di controllo di versione distribuito (DVCS). Permette di tenere traccia dello storico delle modifiche apportate durante lo sviluppo del software. È anche comodo per mantenere più linee di sviluppo, generalmente una pulita e funzionante e un'altra work in progress. Ciò consente agli sviluppatori di risolvere eventuali malfunzionamenti, sicuri di avere comunque una versione operati in rilascio.

In Alchemist viene utilizzata una metodologia che prevede che ogni componente del team di sviluppo abbia una copia locale del progetto principale (una *fork*)[4]. I contributi vengono poi reificati in delle *pull request*. Le pull request permettono di revisionare i contributi, chiedere che vengano integrate delle modifiche e aprire discussioni su porzioni di codice.

[1]https://plantuml.com/
[2]https://github.com/kelvindev15/Kotlin2PlantUML
[3]https://search.maven.org/
[4]https://alchemistsimulator.github.io/howtos/development/contributions/

Conventional commits Si tratta di una convenzione sullo stile dei messaggi dei commit[5]. Si tratta di specificare nel messaggio di commit se la modifica trattasi di una funzionalità che viene aggiunta, risoluzione di bug, refactoring, documentazione o semplicemente di un "lavoretto" (come l'aggiornamento delle librerie importate).

fix(gui): *add spacing between overlapping buttons*

Il formato dei convetional commit, rende i messaggi di commit leggibili sia da un umano che da un calcolarore. Questo significa che è possibile automatizzare il rilascio di nuove versioni in base alla rilevanza delle modifiche.

GitHub actions Ogni volta che viene aperta una *pull request* è importante assicurarsi che il contributo non impatti l'integrità del sistema. Alchemist utilizza un servizio di *Continuos Integration* offerto dalla piattaforma GitHub, chiamato GitHub Actions[6]. Tramite un file di configurazione YAML, vengono definiti uno o più workflow. Ad esempio è possibile eseguire all'interno di macchine virtuali, il processo di compilazione e l'esecuzione dei test. È possibile indicare che la corretta esecuzione senza errori di questi workflow debba essere un requisito vincolante alla integrazione del contributo della *pull request* nella linea di sviluppo principale.

2.3.2 Librerie esterne

Per lo sviluppo dell'ambiente fisico è stata utilizzata la libreria **dyn4j**[7]. È una libreria open-source scritta in Java. Contiene gli algoritmi utili per il rilevamento e la risoluzione delle collisioni.

2.3.3 Controllo di qualità del software

Alchemist, nell'ambito del controllo della qualità, fa uso di strumenti che analizzano il codice con l'obiettivo di segnalare potenziali bug, porzioni di codice duplicato, possibili miglioramenti. Tra gli strumenti utilizzati in Alchemist troviamo:

- **detekt**: è uno strumento che effettua un'analisi statica del codice. Permette di definire un set di regole, come la massima complessità di un metodo o interfaccia, formattazione del codice e commenti e molto altro.
- **klint**: è software *linter* per Kotlin. Un linter analizza il codice sorgente per segnalare errori di programmazione, bug, errori stilistici e costrutti sospetti.

[5]https://www.conventionalcommits.org/en/v1.0.0/
[6]https://github.com/features/actions
[7]https://dyn4j.org/

- **spotbugs**: analizza il *bytecode* Java e rileva possibili bug classificandoli in categorie per dare un'idea sul possibile impatto sul software
- **checkstyle**: è uno strumento che analizza la struttura del codice verificando che rispetti un determinato stile di codifica (ogni linguaggio ha le sue convenzioni).
- **pmd**: è in grado di rilevare tipici errori di programmazione come variabili non utilizzate, creazioni non necessarie di oggetti e cosi via.
- **cpd**: permette di rilevare porzioni di codice duplicato.

Capitolo 3

Valutazione qualitativa

3.1 Simulazione di un'evacuazione di folla in Piazza San Carlo, Torino

Per evidenziare le potenzialità di quanto è stato introdotto con questo contributo, viene di seguito presentato un tentativo di riproduzione di ciò che accadde il 3 giugno 2017 in Piazza San Carlo a Torino, in occasione della finale di Champions League.

3.1.1 Descrizione della simulazione

6200 pedoni di cui 3000 adulti di genere maschile, 3000 adulti di genere femminile, 100 adolescenti di genere maschile, e 100 adolescenti di genere femminile, sono stati collocati in una riproduzione di Piazza San Carlo. Sono state incluse, con approssimazione, ostacoli rappresentanti le transenne. Per questa simulazione, dato che il simulatore non supporta ancora gli ostacoli mobili, le transenne sono state rappresentate come ostacoli immobili. È stata poi inserita una fonte di pericolo nella zona in cui si ritiene sia partita l'isteria di massa[1]. In Figura 3.1 tale zona viene evidenziata in blu.

3.1.2 Risultati ottenuti

Come è possibile osservare nei fotogrammi salienti della simulazione in Figura 3.2, nel momento in cui il pericolo viene percepito dai pedoni intorno all'area in cui è stato collocato, questi iniziano a fuggire. Per contagio sociale [23], anche i pedoni che osservano la fuga iniziano a fuggire. Nell'ultimo fotogramma riportato, è possibile osservare l'effetto *veloce è lento* [10] citato in 1.1.6

[1] `https://www.ilpost.it/2017/06/04/piazza-san-carlo-torino-juventus-real-madrid/`

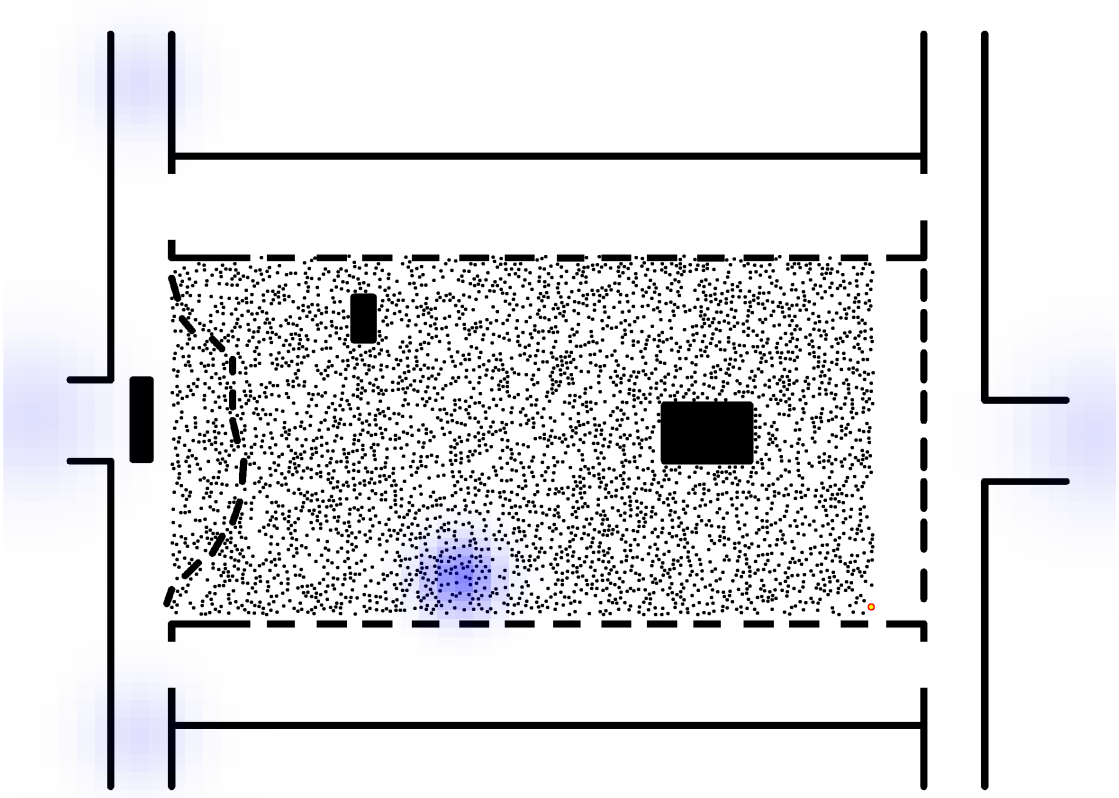

Figura 3.1: Configurazione iniziale della simulazione

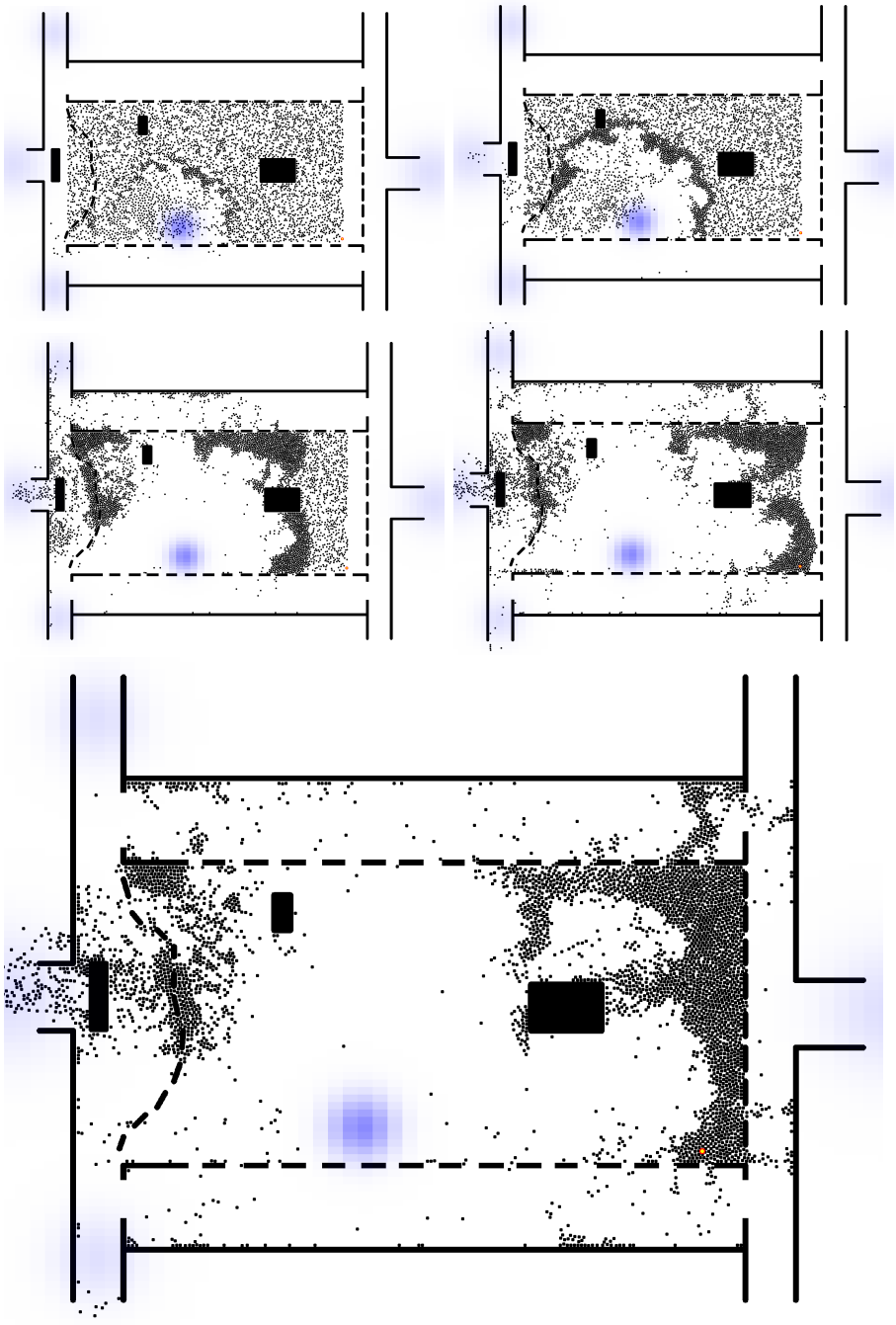

Figura 3.2: Fotogrammi salienti della simulazione

Confronto con la realtà Sebbene la simulazione è stata condotta con un inferiore numero di pedoni rispetto alla realtà, confrontando il risultato della simulazione con i video che riprendono l'accaduto disponibili in rete[2],[3], è possibile trovare una corrispondenza tra ciò che si osserva nella simulazione e ciò che è realmente accaduto. In particolare, un effetto interessante è l'*onda* formatasi per via delle spinte dei pedoni in fuga verso coloro che non hanno direttamente percepito il pericolo.

[2] `https://www.youtube.com/watch?v=FJxn7qs55Aw`
[3] `https://www.youtube.com/watch?v=yuqcNgcgzIA`

Capitolo 4

Conclusioni

4.1 Conlusioni

Obiettivo di questo lavoro era d'introdurre elementi di micro-interazione fisica nel simulatore Alchemist. Si è partiti da un'analisi di ciò che era già presente nel simulatore. Da questa analisi si è pensato che prima di partire fosse necessario intervento strutturale nel simulatore per migliorare la manutenibilità del codice. Infatti l'utilizzo dell'ereditarietà come meccanismo per il riuso del codice, rischiava di limitare e rendere fragile l'architettura. Si è quindi proceduto con un refactoring dell'architettura dei nodi introducendo il concetto di *proprietà* del nodo, sfruttando il meccanismo di *composizione*. Successivamente proceduto con l'introduzione degli elementi d'interazione fisica ispirandosi al modello *HiDAC* di Pelechano et. al [26]. Infine, come valutazione qualitativa del lavoro svolto, è stata costruita una simulazione con l'obiettivo di riprodurre la tragedia che avuto luogo il 3 giugno 2017 in Piazza San Carlo a Torino.

4.2 Sviluppi futuri

Per attribuire ulteriore realismo alle simulazioni, si potrebbe:

- **Attribuire una forma più realistica ai pedoni**: piuttosto che attribuire un'area circolare uguale per tutti i pedoni, si potrebbe differenziarle ad esempio in base all'età.

- **Predisporre il simulatore per includere ostacoli mobili**: queste possono essere ad esempio bottiglie lasciate a terra che nel caos di un'evacuazione potrebbe venir lanciata in giro, facendo inciampare i pedoni.

- **Considerare non solo i pedoni caduti, ma anche i feriti e i deceduti**: Così che si possano analizzare e individuare quelle che sono le cause di questi incidenti.
- **Tenere in considerazione le differenze culturali**: il modo in cui una persona percepisce un pericolo dipende dal contesto in cui vive e dalle sue abitudini.

Bibliografia

[1] Eduard Babulak and Ming Wang. Discrete event simulation: State of the art. *International Journal of Online Engineering (iJOE)*, 4:60–63, 01 2008.

[2] Joshua Bloch. *Effective Java*. Addison-Wesley, Boston, MA, 3 edition, 2018.

[3] GE Bradley. A proposed mathematical model for computer prediction of crowd movements and their associated risks. In *Proceedings of the International Conference on Engineering for Crowd Safety*, pages 303–311. Elsevier Publishing Company London, 1993.

[4] Volkhard Buchholtz and Thorsten Pöschel. In: De wolf, p. grassberger (eds.)"friction, arching, contact dynamics" world scientific (singapore, 1997) p. 265-273.

[5] Dorine Duives, Winnie Daamen, and Serge Hoogendoorn. State-of-the-art crowd motion simulation models. *Transportation Research Part C: Emerging Technologies*, 37:193–209, 12 2013.

[6] Dominic Elliott and Denis Smith. Football stadia disasters in the united kingdom: learning from tragedy? *Industrial & Environmental Crisis Quarterly*, 7(3):205–229, 1993.

[7] George S Fishman. Principles of discrete event simulation.[book review]. 1978.

[8] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of theoretical physics*, 21(3):219–253, 1982.

[9] Michael A Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The journal of physical chemistry A*, 104(9):1876–1889, 2000.

[10] Dirk Helbing, Illés Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[11] Dirk Helbing, Motonari Isobe, Takashi Nagatani, and Kouhei Takimoto. Lattice gas simulation of experimentally studied evacuation dynamics. *Physical review E*, 67(6):067101, 2003.

[12] Dirk Helbing and Anders Johansson. Pedestrian, crowd, and evacuation dynamics. *arXiv preprint arXiv:1309.1609*, 2013.

[13] Dirk Helbing, Anders Johansson, and Habib Zein Al-Abideen. Dynamics of crowd disasters: An empirical study. *Physical review E*, 75(4):046109, 2007.

[14] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[15] Dirk Helbing and Pratik Mukerji. Crowd disasters as systemic failures: analysis of the love parade disaster. *EPJ Data Science*, 1(1):1–40, 2012.

[16] LF Henderson. The statistics of crowd fluids. *nature*, 229(5284):381–383, 1971.

[17] Taira Itoh and Takashi Nagatani. Optimal admission time for shifting the audience. *Physica A: Statistical Mechanics and its Applications*, 313(3-4):695–708, 2002.

[18] Eugene M Izhikevich, John H Conway, and Anil Seth. Game of life. *Scholarpedia*, 10(6):1816, 2015.

[19] Dmitry Jemerov and Svetlana Isakova. *Kotlin in action*. Simon and Schuster, 2017.

[20] Edward V. Krick. *An Introduction to Engineering: Methods, Concepts, and Issues*. John Wiley Sons, 1976.

[21] Gustave Le Bon, Gina Villa, and Piero Melograni. *Psicologia delle folle*. Longanesi, 1996.

[22] Mikaila Mariel Lemonik Arthur. Emergent norm theory. *The Wiley-Blackwell Encyclopedia of Social and Political Movements*, 2013.

[23] Diego Mazzieri. *Progettazione e implementazione di agenti cognitivi per simulazioni di evacuazioni di folle in Alchemist*. PhD thesis.

[24] Clark McPhail. *The myth of the madding crowd*. Routledge, 2017.

[25] Ryoichi Nagai, Masahiro Fukamachi, and Takashi Nagatani. Evacuation of crawlers and walkers from corridor through an exit. *Physica A: Statistical Mechanics and its Applications*, 367:449–460, 2006.

[26] Nuria Pelechano, Jan M Allbeck, and Norman I Badler. Controlling individual agents in high-density crowd simulation. 2007.

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

[28] Craig W Reynolds et al. Steering behaviors for autonomous characters. In *Game developers conference*, volume 1999, pages 763–782. Citeseer, 1999.

[29] Gerald H. Risto and Hans J. Herrmann@f. Density patterns in two-dimensional hoppers. *Phys. Rev. E*, 50:R5–R8, Jul 1994.

[30] Lee Soomaroo and Virginia Murray. Disasters at mass gatherings: lessons from history. *PLoS currents*, 4, 2012.

[31] Kouhei Takimoto and Takashi Nagatani. Spatio-temporal distribution of escape time in evacuation process. *Physica A: Statistical Mechanics and its Applications*, 320:611–621, 2003.

[32] Jan Treur. *Network-Oriented Modeling and Its Conceptual Foundations*, pages 3–33. Springer International Publishing, Cham, 2016.

[33] John von Neumann and Arthur W. Burks. Theory of self reproducing automata. 1966.

[34] C Wal, Daniel Formolo, and Tibor Bosse. An agent-based evacuation model with social contagion mechanisms and cultural factors. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 620–627. Springer, 2017.

[35] Zheng Xiaoping, Tingkuan Zhong, and Mengting Liu. Modeling crowd evacuation of a building based on seven methodological approaches. *Building and Environment*, 44:437–445, 03 2009.

[36] Weifeng Yuan and Kang Hai Tan. An evacuation model using cellular automata. *Physica A: Statistical Mechanics and its Applications*, 384(2):549–566, 2007.

[37] Kathryn M Zeitz, Heather M Tan, M Grief, PC Couns, and Christopher J Zeitz. Crowd behavior at mass gatherings: a literature review. *Prehospital and disaster medicine*, 24(1):32–38, 2009.

[38] Xiaoping Zheng, Tingkuan Zhong, and Mengting Liu. Modeling crowd evacuation of a building based on seven methodological approaches. *Building and environment*, 44(3):437–445, 2009.

Ringraziamenti

La corsa è giunta al termine. Sono stati tre anni intensi, ricchi di gioie e soddisfazioni ma anche di momenti di sconforto e delusioni. Fortunatamente questo è un percorso che non ho affrontato da solo. È dunque arrivato il momento di ringraziare coloro che mi hanno accompagnato durante questa splendida avventura.

Parto dal ringraziare mia Mamma, una donna forte, che mi ha sempre messo al primo posto e non si è mai sottratta ad alcun sacrificio. Ringrazio la famiglia Bartoloni, una seconda famiglia, che da oltre dieci anni c'è sempre stata e mi ha sempre sostenuto. Ringrazio Filippo Pesaresi che mi ha aiutato nei momenti difficili di questo percorso e Don Maurizio che con semplici parole è stato capace di motivarmi e darmi la determinazione. Grazie anche agli amici di Castelbellino e dintorni con cui, ogni volta che torno a casa, trascorro bei momenti in spensieratezza.

Poi ci sono le persone che ho conosciuto qui a Cesena e con le quali ho trascorso i momenti migliori. Sono *Matte*, *Lore*, *Ema*, *Mala*, *Paga*, *Buiz*, e gli amici di *zucchero-sintattico*, *Ale*, *Andru*, *Dj*, *Gigi*, *Gus*, *Ste*, *Tommi*.

Ringrazio i docenti del corso. Una menzione va alla prof.ssa Lazzaro per i momenti di umanità regalati in questi anni. Infine ringrazio infinitamente il prof. Pianini, che mi ha sopportato, supportato, e senza il quale questo lavoro difficilmente si sarebbe portato a termine.

Appendice A

File di simulazione

```yaml
incarnation: protelis

environment:
  type: EnvironmentWithDynamics
  parameters: [../sanCarloSquare.png, 0.08, -100, -76]

variables:
  danger: &danger
    formula: "\"danger\""
  exit1: &exit1
    formula: "\"exit1\""
  exit2: &exit2
    formula: "\"exit2\""
  exit3: &exit3
    formula: "\"exit3\""
  exit4: &exit4
    formula: "\"exit4\""

layers:
  - type: BidimensionalGaussianLayer
    molecule: *danger
    parameters: [-21, -33, 200, 5]
  - type: BidimensionalGaussianLayer
    molecule: *exit1
    parameters: [-105, -1.5, 50, 10]
  - type: BidimensionalGaussianLayer
    molecule: *exit2
    parameters: [-84, -64, 50, 7]
```

```
29    - type: BidimensionalGaussianLayer
30      molecule: *exit3
31      parameters: [-84, 64, 50, 7]
32    - type: BidimensionalGaussianLayer
33      molecule: *exit4
34      parameters: [104, -5, 50, 10]
35
36  _behavior: &behavior
37    - time-distribution:
38        type: DiracComb
39        parameters: [1.0]
40      type: CognitiveBehavior
41    - time-distribution:
42        type: DiracComb
43        parameters: [1.0]
44      type: PhysicalBlendedSteering
45      actions:
46        - type: CognitiveAgentAvoidLayer
47          parameters: [*danger]
48        - type: CognitiveAgentFollowLayer
49          parameters: [*exit1]
50        - type: CognitiveAgentFollowLayer
51          parameters: [*exit2]
52        - type: CognitiveAgentFollowLayer
53          parameters: [*exit3]
54        - type: CognitiveAgentFollowLayer
55          parameters: [*exit4]
56        - type: CognitiveAgentObstacleAvoidance
57          parameters: [6]
58      conditions:
59        - type: WantToEscape
60
61  _adult_deployment: &adult_males
62    - type: Rectangle
63      parameters: [3000, -78.0, -40.0, 138.0, 70.56]
64      properties:
65        - type: Human
66          parameters: ["adult", "male"]
67        - type: Cognitive2D
68          parameters: [*danger]
```

```
      - type: Perceptive2D
      - type: CognitivePedestrian
      - type: PhysicalPedestrian2D
      - type: CircularArea
    programs: *behavior

_child_deployment: &child_males
  - <<: *adult_males
    parameters: [100, -78.0, -40.0, 138.0, 70.56]
    properties:
      - type: Human
        parameters: ["child", "male"]
      - type: Cognitive2D
        parameters: [*danger]
      - type: Perceptive2D
      - type: CognitivePedestrian
      - type: PhysicalPedestrian2D
      - type: CircularArea
    programs: *behavior

deployments:
  - *adult_males
  - <<: *adult_males
    properties:
      - type: Human
        parameters: ["adult", "female"]
      - type: Cognitive2D
        parameters: [*danger]
      - type: Perceptive2D
      - type: CognitivePedestrian
      - type: PhysicalPedestrian2D
      - type: CircularArea
  - <<: *child_males
  - <<: *child_males
    properties:
      - type: Human
        parameters: ["child", "female"]
      - type: Cognitive2D
        parameters: [*danger]
      - type: Perceptive2D
```

```
109         - type: CognitivePedestrian
110         - type: PhysicalPedestrian2D
111         - type: CircularArea
```