UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI SCIENZE

Corso di Laurea in Informatica

Crowd Simulator: un software per la raccolta dati e la simulazione di pedoni in un ambiente virtuale complesso

Relatore	Candidato
Prof. Diego Angelo Gaetano	Gian Mario Marongiu
Reforgiato Recupero	60/61/65850

ANNO ACCADEMICO 2021-2022

INDICE

INDICE DELLE FIGURE

INTRODUZIONE

La simulazione del movimento delle folle di pedoni è un argomento che ha destato l'interesse di diversi campi di ricerca. Questo campo di studi ha iniziato ad essere oggetto di forte attenzione da parte dei ricercatori a partire dagli anni ‘90, sebbene l’interesse per le macchine autonome e la teoria del controllo affondino le loro radici negli anni ‘40, come descritto da Norbert Wiener nel suo libro *Cybernetics, or Control and communication in the Animal and the Machine* [1]. Negli ultimi anni, sono state sviluppate molte tecniche per la rappresentazione verosimile dei movimenti delle folle, sono stati prodotti software di simulazione grafica, sistemi di validazione e valutazione di tali software, nonché indagini e libri sugli approcci e sull’evoluzione di questo ambito di studi. Con il passare del tempo, tale ricerca ha trovato applicazione in svariati campi, come la computer grafica, videogiochi, fisica statistica, robotica, ma anche realtà virtuale, scienze sociali e dinamiche di evacuazione in situazioni di emergenza.

Il presente elaborato, in questo contesto, si propone in qualità di resoconto del lavoro svolto in collaborazione con la Kingston University durante le 375 ore di tirocinio facenti parte del regolare programma di studio del C.d.L. in Informatica dell’Università degli Studi di Cagliari. Il tirocinio è avvenuto in collaborazione con un gruppo di ricercatori interessati al campo di studi sopra citato ed è stato organizzato alternando sessioni di codifica a meeting di confronto sul prodotto del lavoro, ossia una GUI (Graphic User Interface) suddivisa in tre sezioni: la prima consente di visualizzare la simulazione di una folla di pedoni all’interno di un ambiente virtuale bidimensionale; la seconda permette di inserire i parametri che definiscono le caratteristiche dell’ambiente di simulazione; la terza mostra all’utente le caratteristiche delle entità attive nell’ambiente virtuale. L’interfaccia prodotta prende il nome di *Crowd Simulator* e la versione presentata in questo elaborato contiene al suo interno un solo algoritmo di movimento dei pedoni, sebbene nasca con l’intento di contenerne due, tra i quali sia possibile scegliere in fase di inserimento dei parametri della simulazione. Il primo algoritmo, quello attualmente presente nel simulatore, è una versione rivisitata del noto sistema di *flocking* di Craig Reynolds [2]; il secondo è un algoritmo innovativo sviluppato in parallelo al simulatore, la sua assenza all’interno del programma è dovuta al fatto che, durante la produzione del presente elaborato, risulta ancora in fase di studio e miglioramento.

Lo scopo del simulatore è la raccolta di dati inerenti allo spostamento dei pedoni rappresentati. A prescindere dall’algoritmo di movimento utilizzato, a fine simulazione il programma genera un file in sintassi JSON (JavaScript Object Notation) contenente tutte le informazioni necessarie alla ricostruzione dello spostamento dei pedoni da inizio a fine simulazione, nonché quelle necessarie alla rappresentazione degli altri elementi presenti nell’ambiente virtuale, come ostacoli o luoghi di interesse. Grazie alle informazioni che il simulatore fornisce si possono effettuare dei test comparativi o qualitativi, eventualmente utilizzando un sistema o un framework di convalida e valutazione dell’accuratezza dei movimenti. Considerando il suo scopo finale, il programma presentato in questo elaborato è una versione beta, che tuttavia permette una trattazione completa in merito al suo funzionamento e ai suoi sviluppi futuri.

Il presente elaborato è suddiviso in tre sezioni principali. Il primo capitolo si apre con l’introduzione di concetti e terminologie utili ai fini della trattazione, per poi descrivere il panorama scientifico generale, che si concentra sulla simulazione di animali, particelle, pedoni e, in generale, ogni entità del mondo reale che possa venire simulata in un ambiente virtuale più o meno complesso. In questa sezione si descrivono la nascita e gli scopi iniziali delle indagini in questo ambito, per poi analizzare la cronologia evolutiva degli studi incentrati solamente sulla simulazione di pedoni, con una suddivisione temporale che rispecchia le varie fasi

attraverso le quali la ricerca trova il suo sviluppo, ognuna con obiettivi, metodi e tecniche proprie. Il capitolo si conclude con un'introduzione alle tecniche di simulazione principali e alla terminologia specifica di questo settore, per poi fornire una breve introduzione alla sezione successiva. Nel secondo capitolo viene analizzato nel dettaglio il programma *Crowd Simulator*, si descrive l'ambiente di simulazione, con regole, entità e caratteristiche. In seguito, si propone una breve panoramica sul linguaggio e sul framework utilizzati per la codifica, evidenziando le caratteristiche del framework utili alla successiva descrizione dell'interfaccia del programma. Il capitolo si conclude quindi con la descrizione delle funzionalità offerte dall'interfaccia e con l'introduzione alla sezione successiva. Il terzo capitolo scende nel dettaglio degli algoritmi di movimento utilizzati all'interno di *Crowd Simulator*. Questa sezione si apre con la descrizione dell'algoritmo di *flocking* originale di cui si utilizzano i concetti di coesione, separazione e allineamento, per poi analizzare i due algoritmi di movimento già citati. Questo terzo capitolo è fortemente collegato a una sezione aggiuntiva dell'elaborato: l'Appendice A. All'interno dell'appendice sono contenuti i dettagli implementativi di entrambi gli algoritmi di movimento, nonché le caratteristiche hardware/software del sistema utilizzato durante l'intero processo di codifica.

Infine, nelle conclusioni si elencano le proposte di miglioramento di *Crowd Simulator*, i modi in cui verrà esteso e aggiornato. Si effettua poi un breve riepilogo dei temi studiati tra il 2020 e 2022 e delle considerazioni su come ci si aspetta che evolverà il panorama della ricerca nei prossimi anni. In ultimo, un breve commento sul ruolo di *Crowd Simulator* in questo contesto.

1 CAMPO DI STUDI E STATO DELL'ARTE

Questo capitolo propone una panoramica sulla ricerca nell'ambito della simulazione di entità dinamiche all'interno di un ambiente virtuale complesso. Il primo paragrafo costituisce una breve introduzione al tema, si espongono i concetti di *steering behavior* e di entità autonoma, collocandoli all'interno del contesto di studi analizzato. Il paragrafo 1.2. fornisce alcune informazioni sulla ricerca nel campo della simulazione di generiche entità autonome, proponendo alcuni esempi di campi di applicazione; infine, nel paragrafo 1.3 si scende nello specifico dell'ambito del campo di studi che si occupa della simulazione di folle di pedoni, con un occhio di riguardo per la nascita degli studi e la loro evoluzione fino ai giorni nostri, per poi descrivere le principali caratteristiche che le simulazioni di pedoni possono presentare.

1.1 Entità autonome, *steering behavior* e ambiente di simulazione

In questo elaborato si utilizzerà il concetto di pedone con l'accezione di "entità autonoma", ossia un elemento simulato sul modello di un essere vivente del mondo reale. Un'entità autonoma deve essere in grado di adottare un comportamento verosimile sulla base del contesto in cui è inserita e delle caratteristiche dell'ambiente circostante. I movimenti di una entità autonoma, per poter essere considerata tale, prescindono dall'azione umana e si basano solamente sugli algoritmi che lo governano e dalle caratteristiche dell'ambiente di simulazione. Non sono considerabili entità autonome i personaggi di una produzione animata, in quanto le loro azioni sono frutto di una sceneggiatura, e non lo sono neanche gli avatar in un gioco di realtà virtuale, in quanto le loro azioni sono governate dal giocatore che ne comanda gli spostamenti. Un esempio di entità autonoma potrebbe essere un NPC (Non-Player Character), ossia un personaggio di un videogioco che non è comandato dal videogiocatore. Un NPC è considerabile un'entità autonoma da momento in cui può essere inserito in un contesto in cui si trova a interagire con un ambiente circostante (da solo o con un gruppo di suoi simili) e i suoi spostamenti saranno determinati tramite un algoritmo che analizza in tempo reale l'ambiente in cui si trova, producendo degli output coerenti con il suo ruolo all'interno di quell'ambiente.

Nel corso della trattazione si utilizzerà il termine *steering behavior* per riferirsi a quell'insieme di azioni di un'entità autonoma volte allo spostamento da un punto *A* ad un punto *B*, che prevedono un'attività psicofisica e un processo decisionale simulati. Il termine *steering behavior* venne utilizzato da uno dei capisaldi di questa disciplina: Craig Reynolds; utilizzando questo termine si descrivevano degli algoritmi di movimento di entità autonome, questi ultimi non implementano le regole della cinematica, bensì della dinamica dei corpi simulati: il movimento di ogni entità sarà quindi basato su forze fisiche quali la velocità, l'accelerazione, la posizione e in generale tutti gli elementi necessari ad applicare le regole fisiche che stabiliscono il moto di un corpo secondo la dinamica classica. Gli algoritmi del professor Reynolds verranno analizzati più nel dettaglio all'interno del capitolo 3. Nonostante le caratteristiche precise che il professor Reynolds assegnava al concetto di *steering behavior*, nella ricerca scientifica di questo campo di studi si usa spesso questa terminologia per riferirsi a un generico algoritmo di calcolo del movimento di entità autonome.

La struttura che governa i comportamenti delle entità autonome, così come sono intesi all'interno di questo elaborato, è mostrata in Fig. 1.1, dove viene schematizzato il rapporto di subordinazione tra gli elementi che influenzano il movimento di un'entità autonoma. Ogni freccia in figura può essere interpretata come una relazione di dipendenza; dunque, appare chiaro come le strategie di gruppo, ma anche quelle individuali siano influenzate innanzitutto dalla struttura del mondo esterno, il quale può contenere obiettivi, ostacoli o altri elementi che

rivestono un qualunque ruolo durante il calcolo dei processi decisionali simulati. Detto questo, gli *steering behavior* sono l'unione di dinamiche di gruppo e individuali, le prime sono facoltative, poiché, come si è già detto, un'entità autonoma deve poter esistere anche in assenza di suoi simili, ma, allo stesso tempo, in caso ce ne siano non può prescindere dalla loro presenza.

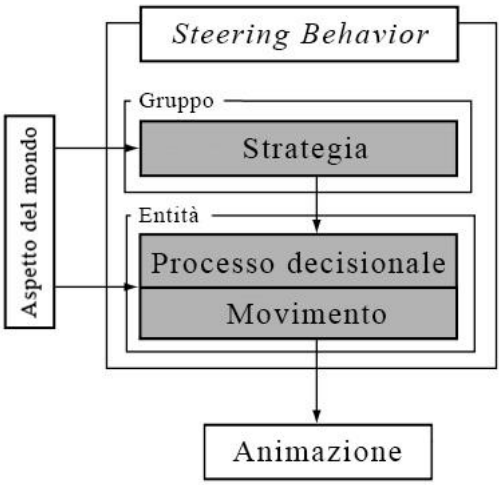

Fig. 1.1: struttura degli *steering behavior* e come questi si relazionano all'ambiente circostante

In ultimo, questi stessi *steering behavior* influenzano l'effettiva rappresentazione grafica delle entità simulate, che sarà compito di un processo di animazione specifico. Questi stessi processi di animazione andranno a rappresentare lo spazio nel quale le entità risiedono utilizzando tecniche di rendering.

1.2 Simulazione di entità autonome

Gli *steering behavior* di entità autonome sono stati, e sono ancora oggi, oggetto di studi da parte di diversi ambiti scientifici. Nel 1999 Craig Reynolds [3] descriveva come l'attenzione per tale campo di studi fosse già al tempo distribuita principalmente tra tre ambiti di interesse: robotica, intelligenza artificiale e vita artificiale. Il lavoro svolto in questi tre campi convergeva spesso in prodotti di animazione o videogiochi, all'interno dei quali venivano simulati un'ampia gamma di entità, principalmente umanoidi e animali. Negli studi che trattavano il movimento di esseri umanoidi era spesso necessario estendere il campo di interesse anche ad altri ambiti come la psicologia e le scienze comportamentali, ad esempio, Ronald C. Arkin [4] fornisce spunti per la progettazione di un sistema di controllo basato su schemi motori per robot mobili, attingendo da lavori inerenti all'ambito delle neuroscienze. Anche quando le entità simulate rappresentavano animali è stato necessario l'appoggio alla biologia e all'etologia, in questo senso, un esempio è il lavoro del professor Ron Arkin [5] [6] [7], concentrato sull'applicazione degli *steering behavior* a robot mobili ispirandosi a preesistenti ricerche etologiche.

Esistono molte ricerche simili a quelle appena citate, un elemento che spesso le accomuna è il peso e la considerazione date ai modelli di percezione-azione delle entità rappresentate, ognuna di queste deve essere in grado di simulare un comportamento improvvisato, che dipende da caratteristiche dell'ambiente che non possono essere previste, dunque è importante creare entità che analizzino l'ambiente circostante (percezione) e si comportino in una determinata maniera a seconda delle occorrenze (azione).

Uno dei lavori più iconici in questo ambito è sicuramente quello del professor Craig Reynolds, che nel suo articolo *Flocks, herds and schools: A distributed behavioral model* [2], definisce degli algoritmi di *flocking*, ossia algoritmi che simulano il comportamento di uno stormo di uccelli secondo uno schema basato sull'utilizzo di tre forze che descrivono il comportamento delle entità simulate: separazione, coesione e allineamento, che verranno descritte nel dettaglio all'interno del paragrafo 3.1. Grazie al suo lavoro, Reynolds pone delle solide basi per molti studi futuri, i quali si estendono verticalmente approfondendo conoscenze e ricerca negli ambiti già citati, ma anche orizzontalmente, proponendo studi che abbracciano ambiti della scienza sempre più variegati. Con il passare degli anni, infatti, si producono studi riguardanti dinamiche di evacuazione [8], utili per implementare sistemi di sicurezza sempre più affidabili; algoritmi di simulazione per la composizione di installazioni d'arte [9]; studi inerenti il comportamento delle greggi di pecore guidate dai cani da pastore [10], simili a quelli riguardanti la simulazione di gruppi di prede che scappano dai predatori [11]; ricerche che propongono sistemi che predicono lo spostamento delle nuvole [12] in maniera tale da raccogliere dati utili al soppesare domanda e offerta di energia generata tramite pannelli solari. Quelli appena citati sono solamente pochi esempi dei campi di applicazione di questo ambito di studi, ma appare chiaro come gli algoritmi di simulazione delle entità autonome trovino applicazione in svariati campi, anche in alcuni nei quali non sembrino poter essere applicati, un esempio è lo studio di S. Liu et al. [13], che analizza il fenomeno della diffusione di una determinata opinione pubblica sulla rete, considerando il pensiero del singolo individuo come se fosse un'entità autonoma che si muove nello spazio del *World Wide Web*, venendo influenzato da ciò che esso contiene.

1.3 Simulazione di folle

Nel paragrafo 1.1 si è visto come quello della simulazione di entità autonome sia un campo di studi particolarmente vasto, per approfondire lo stato dell'arte in questo ambito d'ora in poi ci si concentrerà su una porzione di questo campo: quella riguardante la simulazione di entità umanoidi, che nel corso della trattazione verranno chiamate pedoni.

1.3.1 La ricerca

Con il passare degli anni sono stati scritti molti articoli in merito alle tecniche di simulazione di folle di pedoni, con un interesse sempre crescente e una trattazione di temi sia ricorrenti che innovativi. Il numero di articoli in questo ambito è cresciuto al punto da rendere possibile anche la nascita di indagini che analizzassero la cronologia delle produzioni scientifiche: alcune di queste indagini si focalizzavano maggiormente su lavori incentrati sull'aspetto dell'analisi delle folle [14] [15] (analisi comportamentale, statistica, ecc.); altre hanno proposto un resoconto più completo, un esempio è quello di M. Azahar et al. [16], che presenta i pionieri nel campo della simulazione di folle organizzati in sequenza temporale fino al 2006, concentrandosi principalmente sulle applicazioni in tempo reale degli studi analizzati e sulle sfide di ogni arco temporale descritto; altre ricerche hanno prodotto un resoconto degli studi che utilizzavano precisi approcci alla simulazione di folle, come il lavoro di K. Ijaz et al. [17] che propone un'analisi degli articoli che trattano il tema della simulazione solamente tramite tecniche ibride. Nel 2021 viene proposto un ennesimo articolo riassuntivo di tutte le ricerche effettuate nel campo della simulazione delle folle, Musse et al. [18] propongono una suddivisione in archi temporali di questa area di studi, ognuno di questi ha le sue caratteristiche peculiari, tematiche di interesse e problemi affrontati. Per effettuare il resoconto degli articoli prodotti si è ricorso

ai motori di ricerca di ACM, IEEE, Elsevier, Springer e Google Scholar, utilizzando le parole chiave: “Crowds” (folle) e “Simulation” (simulazione). Questo processo di selezione ha prodotto migliaia di articoli, dai quali ne sono stati scelti solamente 319 in base all'argomento principale. Questi articoli partono dal 1987 fino ad arrivare al 2021, così come mostrato in Fig. 1.2: Numero di articoli prodotti in relazione al periodo di osservazione..

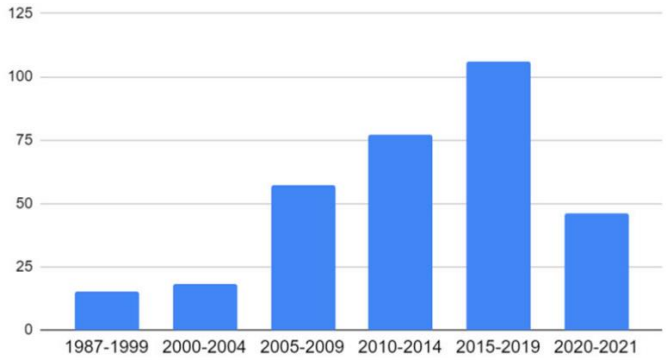

Fig. 1.2: Numero di articoli prodotti in relazione al periodo di osservazione.

Sull'asse orizzontale gli anni, in quello verticale il numero di pubblicazioni.

(Fonte: [18])

Dai risultati di questa analisi si è osservato che gli articoli prodotti prima del 2000 sono per la maggior parte lavori incentrati su “flocks and behaviors” (stormi/branchi e comportamenti), ossia sul lavoro di Craig Reynolds. In questo senso, alcuni dei lavori più iconici sono: quello di Helbing e Molnár [19] che propone un modello basato sulle forze fisiche e sociopsicologiche per descrivere il comportamento della folla in situazioni di panico; quello di Musse e Thalmann [20] che propone il primo metodo di simulazione della folla basato su un controllo gerarchico, con la folla formata da gruppi formati da individui, inoltre, in questo lavoro, aspetti sociologici come la leadership sono stati riformulati matematicamente per essere presi in considerazione nelle decisioni della folla; un secondo articolo di Reynolds stesso [3], che affronta nuovamente l’argomento nel 1999 proponendo gli *steering behavior* come soluzione per modellare entità autonome nell'animazione e nei videogiochi, dotandoli della capacità di spostarsi nel mondo in modo realistico e improvvisato.

Gli articoli prodotti tra il 2000 e il 2004 sono caratterizzati dalla presenza di strutture all’interno della folla simulata, dall’approfondimento dei comportamenti della stessa e dalle connessioni con l’ambiente circostante. Se nel periodo precedente si usava spesso il termine “pedoni” per descrivere le entità simulate, adesso si utilizza di frequente “virtual humans” (umani virtuali). Nel 2000 viene prodotto un articolo [21] che propone un'architettura per simulare umani virtuali in ambienti complessi, i quali racchiudono regole comportamentali che gli umani virtuali possano applicare. In questo periodo si inizia ad osservare l’uso di termini “panic”, “evacuation”, “escape”, a dimostrazione che la ricerca iniziava a focalizzarsi sulle tecniche di evacuazione. Per esempio, Helbing et al. [22] esaminarono il meccanismo di panico e interferenze tra pedoni all’interno di una folla, suggerendo modi pratici di prevenire l’accalcamento in situazioni di pericolo e strategie ottimali per fuggire da una stanza piena di fumo. In questi anni, come si è già detto, ci si concentra anche sulle dinamiche di folla, integrate allo stesso tempo con i metodi decisionali del singolo individuo, Loscos et al. [23] nel 2003

propongono una tecnica che permette la rappresentazione contemporanea di 10000 pedoni tramite un metodo di simulazione che tenesse in considerazione e ottimizzasse comportamenti e reazioni locali e globali dei pedoni.

Gli anni tra il 2005 e il 2009 sono caratterizzati dall'ampia proposta di nuove tecniche di navigazione. Sung et al. [24] descrivono un algoritmo di calcolo del percorso basato su cronoprogrammi probabilistici per navigare all'interno di un ambiente simulato complesso. Altre tecniche proposte coinvolgevano mappe a corridoio [25], o pianificazione gerarchica del percorso [26]; nel 2007 inoltre alcuni ricercatori iniziarono a proporre dei metodi di simulazione *data-driven*,[1] Lerner et al. [27], ad esempio, propongono un metodo basato su osservazioni di reali folle di pedoni: nel loro modello i pedoni simulati prendevano costantemente esempio da pedoni reali, interpretando la situazione in cui si trovavano e ricavando dai dati delle persone reali il comportamento più consono a quel determinato contesto. Inoltre, in questo periodo, Pelechano et al. [28] propongono di incorporare modelli psicologici, ruoli e comunicazione nella simulazione di folle, nonché l'introduzione di un leader [29] che influenzi le azioni del resto della folla. Come dimostrato, questo periodo è stato ricco di nuove idee, la ricerca ha iniziato ad espandersi velocemente, tanto che sono stati pubblicati i primi due libri completamente dedicati alla simulazione di folle, uno di Musse e Thalmann [30] e uno di Pelechano, Allbeck e Badler [31].

Tra il 2010 e il 2014 si consolidano le tecniche di navigazione e prevenzione delle collisioni, e si iniziano a proporre modelli di gestione di personalità e metodi basati sull'implementazione di pedoni che possiedono una visione artificiale. Durupinar et al. [32], nel 2009, creano una tecnica di simulazione di folle in cui ogni pedone ha una personalità basata su apertura mentale, coscienziosità, estroversione, amichevolezza e nevroticismo; ognuna di queste caratteristiche va a influenzare il comportamento finale di ognuna delle entità simulate. Tra i lavori che sviluppavano un sistema di visione sintetica c'è quello di Dutra et al. [33] che esplora un approccio alla prevenzione di collisioni *vision-based* all'interno del quale ad ogni fase della simulazione, una telecamera posta su ciascun pedone permette il rendering dell'ambiente circostante dal suo punto di vista, permettendo all'entità di vedere gli ostacoli sul suo percorso e comportarsi di conseguenza.

Infine, dal 2015 al 2019, continua la produzione di documenti relativi ai metodi *data-driven* e un numero significativo include tecniche di machine learning e previsione statistica. Spesso gli autori estraggono dati dalle folle reali per costruire modelli che descrivano traiettorie efficienti e verosimili. Ahmet et al. [34], ad esempio, propongono un approccio *data-driven* per l'ottimizzazione e la convalida delle simulazioni di folle, estraendo informazioni da video reali. Quello dell'evacuazione è un tema ricorrente in questo periodo, Testa et al. [35], ad esempio, utilizzano ANNs (Artificial Neural Networks) per stimare (non simulare) il tempo di evacuazione di una folla da un ambiente complesso, ottenendo un errore medio del 5% rispetto alle evacuazioni in un ambiente reale di una folla vera. Nel 2017 viene perfino pubblicato un libro intitolato *Simulating Crowds in Egress Scenarios* [36], completamente dedicato alla simulazione di folle di pedoni, con un occhio di riguardo per l'analisi di scenari di emergenza. Oltre alle tecniche di apprendimento automatico, in questo periodo sono state proposti anche degli *authoring-system*,[2] un esempio è il lavoro di Ho et al. [37], dove si propone un sistema che consente la modifica in tempo reale dell'ambiente in cui la folla viene simulata, gli utenti

[1] I movimenti dei pedoni basati sui metodi *data-driven* dipendono da informazioni contenute in dati preesistenti piuttosto che da calcoli svolti in tempo reale.

[2] Gli *autoring-system* sono degli applicativi che consentono la realizzazione di una comunicazione multimediale, articolata e riproducibile su personal computer.

che utilizzano l'applicativo possono disegnare le regioni di spazio attraenti (verso le quali i pedoni dovranno dirigersi) e le regioni non attraenti (che dovranno evitare). Sempre in questo periodo iniziano a venire proposti i primi lavori in realtà virtuale, aumentata e mista, Moussaïd et al. [38], per esempio, propongono uno studio su un ambiente virtuale tridimensionale che fa parte di una piattaforma sperimentale per condurre esperimenti sulle folle utilizzando persone reali.

Dopo il 2020 si continua lo sviluppo di tecniche immersive, orientate alla realtà virtuale e si continua a dare peso alle tecniche di calcolo del movimento tramite machine learning e allo sviluppo di modelli basati sulle emozioni delle entità simulate. In particolare, si propongono alcuni metodi per assistere la mobilità delle persone in scenari di evacuazione di edifici, situazioni di pericolo come un attacco terroristico, strategie per trovare percorsi ottimali e persino metodi per simulare la mobilità umana durante la situazione COVID-19. Alcuni esempi sono lo studio di Mirahadi e McCabe [39] che propongono un modello *real-time* per l'evacuazione di un edificio basata sull'algoritmo di Dijkstra, o l'articolo di Orallo e Martinez [40], in cui si studia un modello per valutare il rischio spaziale e temporale di trasmissione del COVID-19 in funzione del movimento dei pedoni.

1.3.2 Le tecniche di simulazione

Nel precedente paragrafo ci si è incentrati sull'ampia varietà di studi in questo campo senza scendere nei dettagli delle tecniche di movimento dei pedoni e della loro gestione. Parlando dell'intera folla di pedoni, le tecniche che ne definiscono il movimento solitamente si dividono in due grandi famiglie, la prima è quella delle tecniche macroscopiche [41]. Queste ultime hanno un approccio *flow-based*, ossia si concentrano sui comportamenti dell'intera folla come se fosse un'unica entità, sono particolarmente efficienti per simulare in maniera poco dettagliata gli spostamenti di un gran numero di entità; quindi, vengono utilizzate soprattutto negli studi in cui si cerca di fare una stima dello stato della folla in tempo reale senza soffermarsi sull'esatto movimento del singolo. La seconda grande famiglia è quella delle tecniche microscopiche [42], che si concentrano invece sulla gestione del singolo pedone, sono utili per calcolare il percorso che la singola entità deve seguire evitando eventuali ostacoli statici o dinamici presenti nell'ambiente. Questa tecnica implementa un approccio detto *agent-based*, caratterizzato da individui autonomi con un certo grado di intelligenza; ognuno di loro può reagire a ogni situazione in modo autonomo sulla base di un insieme di regole decisionali. Le informazioni utilizzate per decidere un'azione sono ottenute localmente dall'ambiente circostante. Il più delle volte questo approccio viene utilizzato per simulare un comportamento realistico della folla, in quanto il ricercatore ha piena libertà di implementare qualsiasi comportamento.

Le folle di pedoni possono avere determinate caratteristiche. Solitamente si parla di folle omogenee se ogni pedone ha comportamenti e obiettivi molto simili, oppure di folle eterogenee se ogni pedone mantiene un'identità distinta e osservabile. A prescindere dalle sue caratteristiche intrinseche, il calcolo del movimento di un pedone è suddiviso in due approcci distinti: quello che gestisce la navigazione globale e quello che gestisce la navigazione locale. La navigazione globale mira a calcolare un percorso a lungo termine senza collisioni verso un obiettivo, considerando solamente gli ostacoli statici. Al contrario, le tecniche di navigazione locale tengono conto anche del movimento di eventuali ostacoli dinamici e delle altre entità nell'ambiente.

La pianificazione del movimento può a sua volta essere divisa in due ulteriori approcci: centralizzato e disaccoppiato. L'approccio centralizzato considera tutti i pedoni e tratta il sistema risultante come un unico sistema composito dove i possibili movimenti dei singoli

pedoni sono combinati in uno spazio composito in modo che l'algoritmo risultante cerchi una soluzione in questo spazio di configurazione, come se stesse risolvendo un labirinto per ognuno dei pedoni contemporaneamente. Al contrario, l'approccio disaccoppiato procede in modo distribuito tra i vari pedoni, calcolando il percorso della singola entità; perciò, l'algoritmo di calcolo del movimento del singolo pedone ha necessità di essere coordinato con quello degli altri pedoni presenti nella simulazione. Il primo di questi due approcci, quello centralizzato è tendenzialmente meno efficiente rispetto a quello disaccoppiato poiché si basa su un principio molto simile a quello degli algoritmi di *path finding*, ossia di risoluzione dei labirinti, con l'aggravante di dover tenere conto non solo della struttura all'interno della quale trovare il percorso ottimale, ma anche della presenta delle altre entità.

1.4 Conclusioni

Il seguente capitolo fornisce un'introduzione completa al campo di studi d'interesse e un contesto per i temi dei due capitoli successivi. Si è visto come la ricerca è nata e si è evoluta, quali studi sono stati fatti e quali sono le ricadute principali sui vari ambiti di applicazione. Una volta chiaro il contesto in cui si colloca questo elaborato, il capitolo successivo introduce il lavoro che ha preso ispirazione dagli studi di questo ambito, ossia il programma *Crowd Simulator*.

2 *CROWD SIMULATOR*: SPECIFICHE E CODIFICA

L'obiettivo di questo capitolo è presentare il programma *Crowd Simulator* descrivendone funzionalità e aspetto dell'interfaccia. Nel paragrafo 2.1 si descrivono nel dettaglio le entità che il programma gestisce e l'ambiente di simulazione che viene rappresentato. In seguito, si espongono le motivazioni che hanno portato alla selezione del linguaggio di programmazione scelto per la codifica del programma, con una breve citazione alle caratteristiche principali del framework utilizzato. Infine, nel paragrafo 2.3, si utilizzano i concetti esposti in merito al framework per descrivere le sezioni che compongono l'interfaccia e le sue funzionalità.

2.1 L'ambiente di simulazione

Il programma *Crowd Simulator* propone un'interfaccia divisa in sezioni tramite le quali sia possibile definire le caratteristiche di un ambiente virtuale e visualizzare la folla di pedoni che si sposta al suo interno. Nell'ambiente di simulazione, a prescindere dall'algoritmo utilizzato per il movimento dei pedoni, è sempre presente un edificio e delle entità statiche contenute in esso: ostacoli e obiettivi. I pedoni attraversano l'edificio interagiscono con gli elementi in esso contenuti e poi lo abbandonano.

2.1.1 Edificio ed entità statiche

Nell'ambiente di simulazione la prima cosa che viene mostrata a schermo è la struttura dell'edificio all'interno del quale andranno a muoversi i pedoni, così come mostrato in Fig. 2.1. L'edificio è costituito da un ampio corridoio centrale e da un numero di stanze casuale, che può variare da un minimo di sei ad un massimo di dodici. Ogni stanza ha un'unica porta per l'ingresso e l'uscita, mentre l'edificio ha un ingresso e un'uscita separati e collocati ai lati opposti del corridoio centrale.

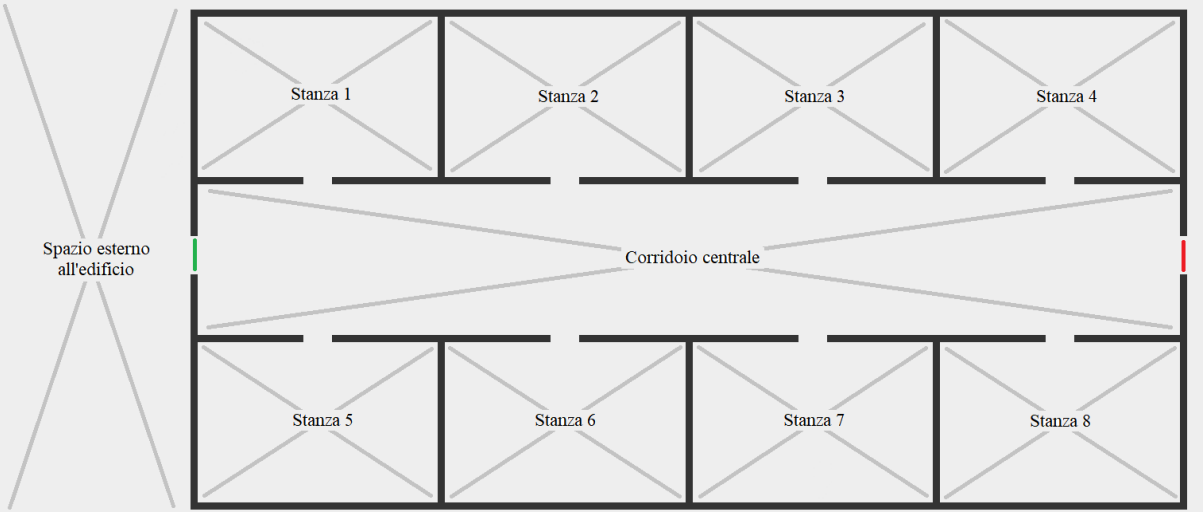

Fig. 2.1: la struttura in nero costituisce l'edificio, in verde la sua porta di entrata e in rosso la sua porta di uscita (ulteriori annotazioni non fanno parte dell'effettiva grafica della simulazione)

All’interno dell’edificio sono collocate due tipologie di entità statiche: gli ostacoli, ovvero gli oggetti che il pedone deve evitare ogni qualvolta vengano incrociati lungo il cammino, in Fig. 2.2 sono rappresentate da quadrati e cerchi neri; gli obiettivi (o punti di interesse), che costituiscono luoghi di interesse e di passaggio per i gruppi di pedoni, che vi effettuano la sosta per un intervallo di tempo arbitrario, per poi andare via e passare al prossimo obiettivo, fino a raggiungere l’uscita dell’edificio. Gli obiettivi, a loro volta, si distinguono in due tipologie: quelli generici e i punti di ristoro, i primi non hanno nessuna proprietà aggiuntiva e sono rappresentati da un quadratino rosso, mentre i secondi, come si vedrà nel paragrafo successivo, sono il luogo in cui i pedoni possono recuperare l’energia spesa durante l’attraversamento dell’edificio e sono rappresentati da un quadratino blu. Le entità statiche (sia ostacoli che obiettivi) hanno un ID identificativo per renderne più semplice l’identificazione all’interno del pannello delle entità attive[3] e possono essere situate in qualsiasi punto interno all’edificio, ma non in prossimità delle porte, così come mostrato in Fig. 2.2.

2.1.2 Folla, gruppi e pedoni

La folla è eterogenea: è composta da un insieme di pedoni con caratteristiche personali e differenti da quelle delle altre entità. Ogni pedone ha un gruppo di appartenenza, una lista di obiettivi da raggiungere prima di uscire dall’edificio, un sesso (maschio o femmina), un’età (bambino, giovane, anziano), un certo livello di energia, una velocità massima.[4] Ogni pedone si sposta lungo l’edificio raggiungendo gli obiettivi assieme al suo gruppo, con il quale deve rimanere coeso, senza però collidere con altri membri.

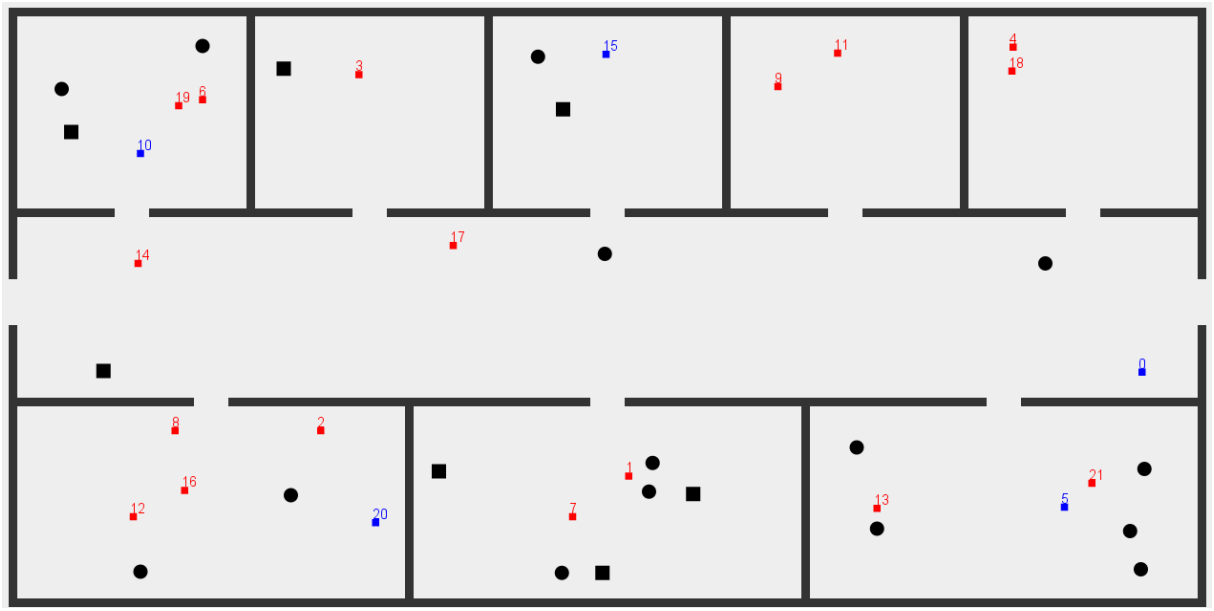

Fig. 2.2: Edificio contenente ostacoli e obiettivi. In nero gli ostacoli, in rosso gli obiettivi e in blu i punti di ristoro

[3] Il pannello delle entità attive mostra un resoconto di tutti gli elementi presenti nella simulazione, il suo funzionamento viene descritto più nel dettaglio nel paragrafo 2.3.

[4] I valori di energia e velocità massima del pedone sono generati casualmente all’interno di un certo intervallo casuale, che varia in funzione dell’età. Tendenzialmente i pedoni più anziani saranno più lenti e avranno meno energia.

Ogni gruppo ha una grandezza casuale all'interno di un determinato intervallo stabilito dall'utente tramite l'interfaccia della GUI, i componenti del gruppo una volta disposti al di fuori dell'edificio inizieranno a muoversi solamente dopo un intervallo di tempo dato, al termine del quale potranno cominciare la loro ricerca dei punti di interesse. La lista di obiettivi è la stessa per ogni membro del gruppo e viene costruita partendo dall'insieme di tutti gli obiettivi disponibili nell'edificio e assegnandoli ad uno ad uno al gruppo secondo una certa probabilità, che varia in base alla grandezza del gruppo stesso in relazione al numero di obiettivi disponibili. Durante l'assegnazione degli obiettivi non si prendono in considerazione i punti di ristoro, poiché questi verranno aggiunti alla lista nel corso della simulazione e solamente all'occorrenza, ossia quando un pedone scende sotto a una certa soglia limite di energia. Quando un pedone deve dirigersi presso un punto di ristoro, vi ci conduce il suo intero gruppo, per poi effettuarvici una sosta fino a quando tutti i membri non avranno nuovamente ripristinato l'energia fino alla sua soglia massima.

I pedoni vengono inizialmente disposti in ordine sparso nello spazio esterno all'edificio, da qui, si riuniscono in modo da poter accedere alla struttura utilizzandone l'ingresso. Chiaramente, pur essendo entità autonome, i pedoni devono rispettare determinate dinamiche di gruppo, ed essere anche capaci di romperle all'occorrenza. In generale, le loro strategie comportamentali variano nel corso della simulazione dipendentemente da una serie di parametri interni (variazioni di stato) o esterni al singolo individuo. Un parametro interno può essere ad esempio la posizione attuale del pedone: se un pedone è all'esterno dell'edificio, prima di dirigersi verso la porta di ingresso è opportuno che si riunisca con il suo gruppo, dunque in questo caso si favorisce la forza di coesione piuttosto che quella che lo indirizzerebbe verso la porta della struttura, allo stesso tempo, la coesione diventa una forza trascurabile se un certo pedone deve uscire velocemente da una stanza all'interno della quale non dovrebbe stare.[5] I parametri esterni sono più numerosi, il principale è l'obiettivo corrente: parametro fondamentale che guida l'intera strategia comportamentale del pedone, determina il verso e la direzione di moto. La strategia appena descritta viene però ignorata nel caso in cui si entri in un contesto nel quale il pedone si ritrova nelle vicinanze di ostacoli: se il pedone ne incontra uno lungo il suo cammino la strategia prioritaria non sarà più raggiungere l'obiettivo, bensì evitare l'ostacolo; muri dell'edificio: i pedoni non possono ignorarli, bensì evitare le collisioni e usare le porte dello stesso; dunque, anche in questo caso la politica di non collisione si impone su quella di raggiungimento del punto di interesse; altri pedoni: ogni pedone deve rimanere vicino agli altri membri del suo gruppo senza collidere con loro o con i membri di altri gruppi.

2.2 *Swing*: Scelta del framework e peculiarità

Crowd Simulator è stato codificato utilizzando il linguaggio Java e il framework *Swing*. Sulla base delle specifiche del programma da implementare, il linguaggio e il framework sono stati scelti sulla base del concetto di semplicità. La scelta del linguaggio è stata influenzata principalmente dalla familiarità con lo stesso e dalla semplicità di fruizione e utilizzo del framework, *Swing* e Java infatti possiedono una buona documentazione ufficiale, completa di esempi di codice eseguibile, casi d'uso, analisi delle criticità e descrizione dei problemi comuni.

[5] Nel corso della simulazione può capitare che un pedone rimanga indietro rispetto al resto del gruppo perché "spinto" dagli altri membri della folla, questa spinta in rari casi porta il pedone a rimanere in una stanza in cui non c'è più nessun obiettivo presso il quale stazionare, mentre il resto del suo gruppo l'ha già lasciata. Appare chiaro che in questa situazione l'obiettivo primario è uscire dalla stanza il prima possibile per riunirsi al resto del gruppo; dunque, la coesione in un contesto del genere diviene una forza secondaria o da ignorare.

Swing è un framework orientato allo sviluppo di interfacce grafiche. Fa parte delle JFC (Java Foundation Classes)[6] e si sviluppa sopra il precedente AWT (Abstract Windowing Toolkit)[7], estendendone i componenti e rendendoli più potenti e flessibili. I programmi scritti utilizzando *Swing* sono completamente indipendenti dalla piattaforma su cui vengono eseguiti e si basano sul paradigma MVC (Model View Controller), che separa la logica di presentazione dei dati dalla logica di gestione degli stessi. I componenti di *Swing*, inoltre, si snodano seguendo una precisa gerarchia, così come mostrato in Fig. 2.3: gerarchia dell'API Java Swing., che ha come biforcazione principale la divisione tra *Container* e *JComponent.*

Un *Container* è un componente che ha l'importante proprietà di poter racchiudere altri elementi al suo interno (che a loro volta possono essere *Container* o *JComponent*). In particolare, in *Crowd Simulator* si è fatto utilizzo di un unico *Frame* e di molteplici *Panel*, i 3 fondamentali sono quelli che costituiscono le tre sezioni dell'interfaccia descritte nel paragrafo 162.3.

Un *JComponent* invece è un qualunque altro widget del framework. Questo insieme include pulsanti (*JButton*), etichette e stringhe (*JLabel*), menù di diverso tipo (*JMenu*) e, in generale, tutti quegli elementi dell'interfaccia con cui è possibile interagire allo scopo di ricevere una risposta da parte del sistema.

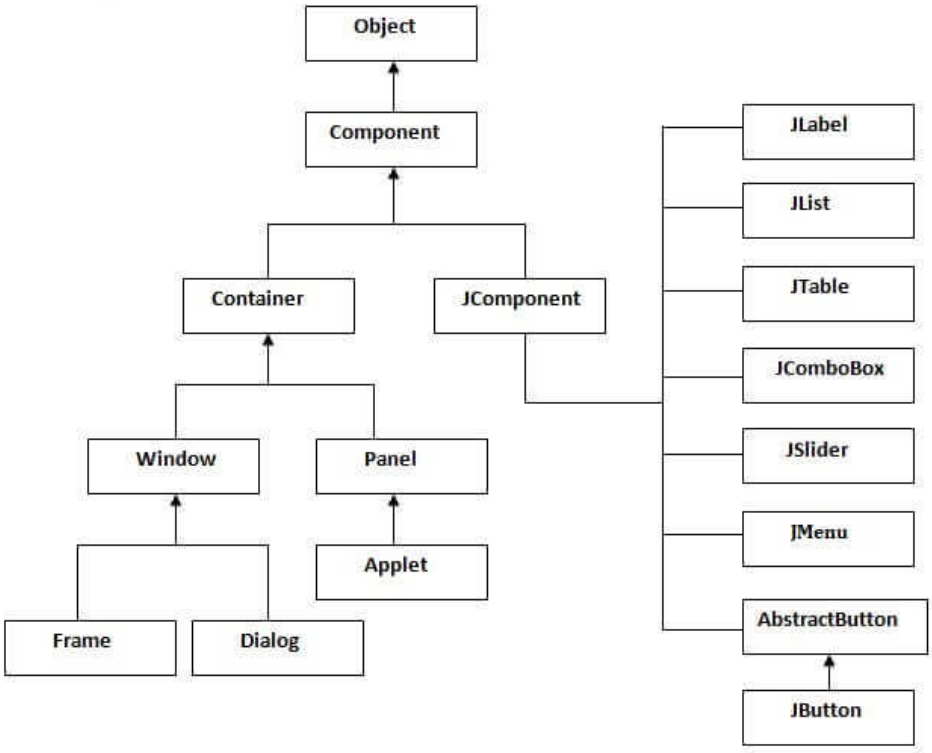

Fig. 2.3: gerarchia dell'API Java Swing.

(Fonte: https://www.javatpoint.com/java-swing)

[6] JFC è un set di componenti (finestre, pannelli e widget in generale) per lo sviluppo di applicazioni desktop
[7] AWT è la libreria Java contenente le classi e le interfacce fondamentali per il rendering grafico.

Le componenti appena descritte costituiscono gli elementi dell'interfaccia grafica con la quale è possibile interagire, tuttavia, la simulazione animata vera e propria fa uso della già citata libreria AWT, in particolare della sua classe java.awt.Graphics2D, la quale fornisce la possibilità di disegnare immagini, forme o stringhe di testo sui pannelli di *Swing*, nonché di gestirne il colore, eventuali trasformazioni, o spostamenti. Il processo di animazione è coadiuvato da un Timer ciclico fornito da *Swing*, al termine del quale vengono cancellati tutti gli elementi della schermata di simulazione e vengono ridisegnati con le posizioni aggiornate, dando così l'illusione del movimento.

2.3 Funzionalità dell'interfaccia

La GUI è interamente contenuta all'interno di un JFrame, che a sua volta si compone di tre JPanel all'interno dei quali sono situate le tre sezioni principali dell'interfaccia. Il primo di questi pannelli è quello che mostra il già descritto ambiente di simulazione, mentre quelli adiacenti contengono tutte le funzionalità dell'interfaccia. Il secondo pannello è denominato *Settings Menu* e si trova nella parte sinistra della GUI. Tramite questa sezione è possibile impostare i parametri della simulazione, come la grandezza della folla, il numero di ostacoli e di obiettivi, il numero di pedoni appartenenti a ogni gruppo. Tramite questa sezione è anche possibile dare il via alla simulazione, metterla in pausa e bloccarla, resettando i parametri di quella corrente, così da poterne avviare una nuova. All'avvio del programma il pannello in questione si presenta come in Fig. 2.4, per avviare la simulazione l'utente deve inserire i parametri negli appositi campi, cliccare sul pulsante "conferma" e poi sul pulsante "play", ossia quello centrale nella parte alta dello schermo. Il valore di ogni parametro inserito deve essere maggiore di 0, in caso contrario compare una finestra di avviso che avvisa l'utente dell'errore commesso. Una volta che la simulazione è avviata, verranno sbloccati anche i pulsanti "pause" (a sinistra di "play") e "stop" (a destra di "play"), che permettono rispettivamente di sospendere e di terminare la simulazione in corso.

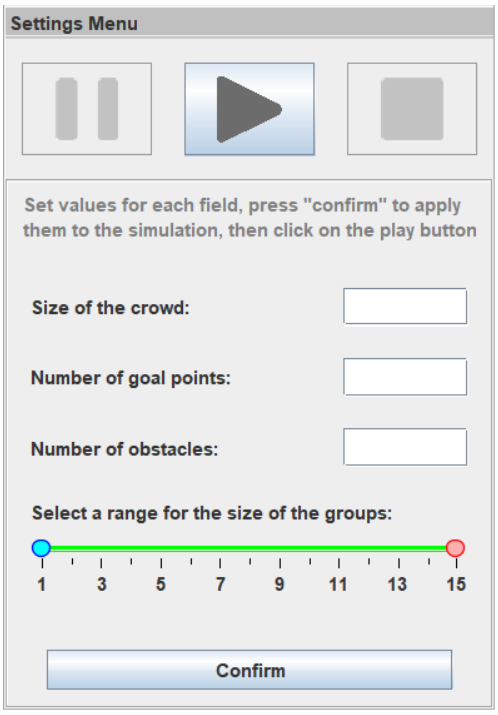

Fig. 2.4: Aspetto del pannello *Settings Menu* prima dell'inizio della simulazione

Infine, il terzo dei tre pannelli si trova nella parte bassa della schermata, non è denominato in nessuna maniera specifica in quanto è diviso a sua volta in tre *tab*, ossia in tre pannelli sovrapposti ed intercambiabili, il primo dei quali mostra le informazioni relative ai pedoni, il secondo contiene le informazioni relative agli ostacoli e il terzo quelle relative agli obiettivi, così come mostrato in Fig. 2.5: pannello delle entità attive prima dell'inizio della simulazione.

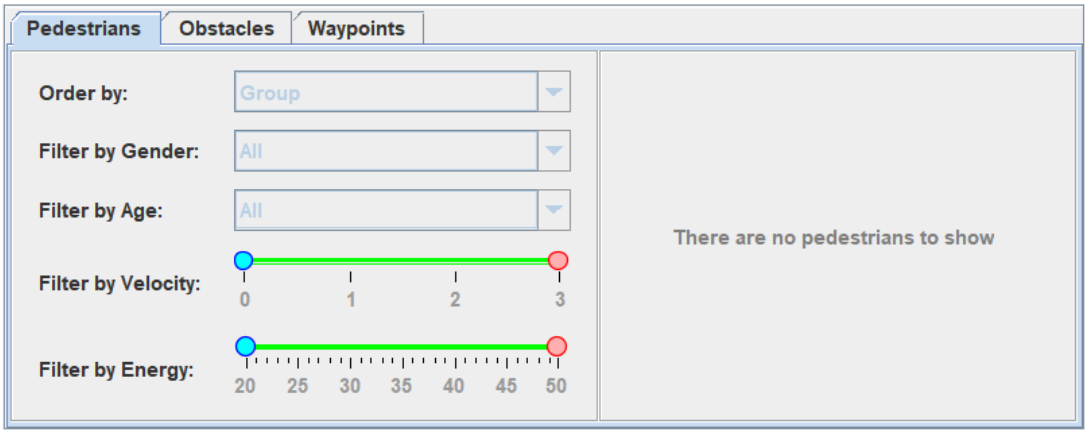

Fig. 2.5: pannello delle entità attive prima dell'inizio della simulazione

In questa sezione è possibile visualizzare i pedoni suddivisi per gruppo di appartenenza, vedere gli obiettivi del gruppo, ordinare e filtrare i pedoni in base ai loro parametri principali (sesso, età, velocità ed energia). Inoltre, il pannello si aggiorna man mano che i dati della simulazione variano, mostrando il consumo e il recupero dell'energia dei pedoni, così come l'obiettivo corrente e quelli non ancora raggiunti. Alle entità statiche non sono associate funzioni di ordinamento o filtraggio, ma solamente un ID univoco che viene riportato sia sul pannello delle entità attive sia, come già mostrato in Fig. 2.2, sulla schermata di simulazione affianco alla relativa entità. Una volta avviata la simulazione, il pannello delle entità attive appare come in Fig. 2.6: aspetto del pannello dei pedoni attivi , le opzioni di filtraggio e ordinamento vengono sbloccate, mentre sul pannello destro vengono mostrati i pedoni organizzati per gruppo.

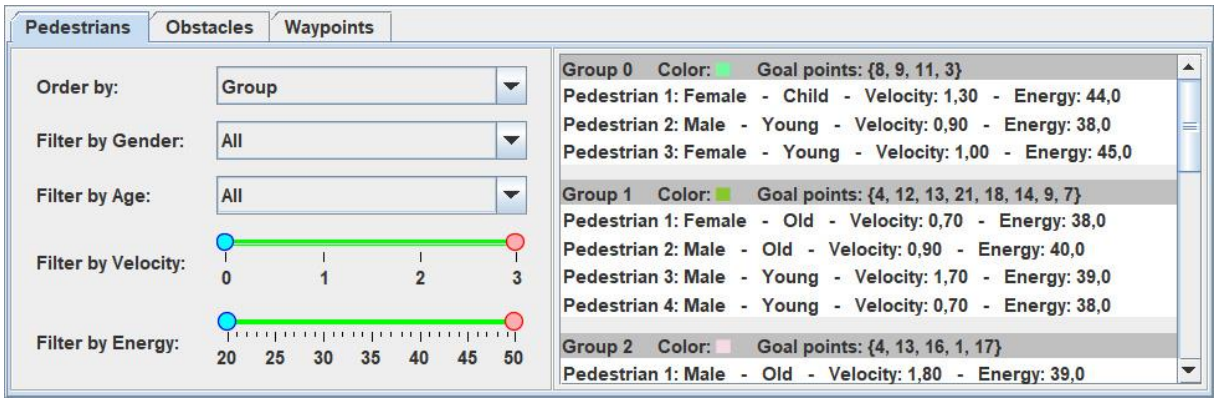

Fig. 2.6: aspetto del pannello dei pedoni attivi dopo aver avviato la simulazione

Nella suddivisione di pedoni per gruppi mostrata in Fig 2.6, a ognuno di loro viene associato un colore, così renderne i membri più facilmente individuabili all'interno dell'ambiente di

simulazione. Inoltre, per ogni gruppo si mostrano gli obiettivi che restano ancora da raggiungere, inserendone gli identificativi tra parentesi graffe e man mano che i pedoni visitano gli obiettivi questi ultimi vengono rimossi dalla lista.

Dopo aver avviato la simulazione i due pannelli delle entità attive rimanenti racchiudono le informazioni che riguardano ostacoli e obiettivi. Essendo elementi statici se ne riporta solamente la posizione e il numero identificativo, così come mostrato in Fig. 2.7.

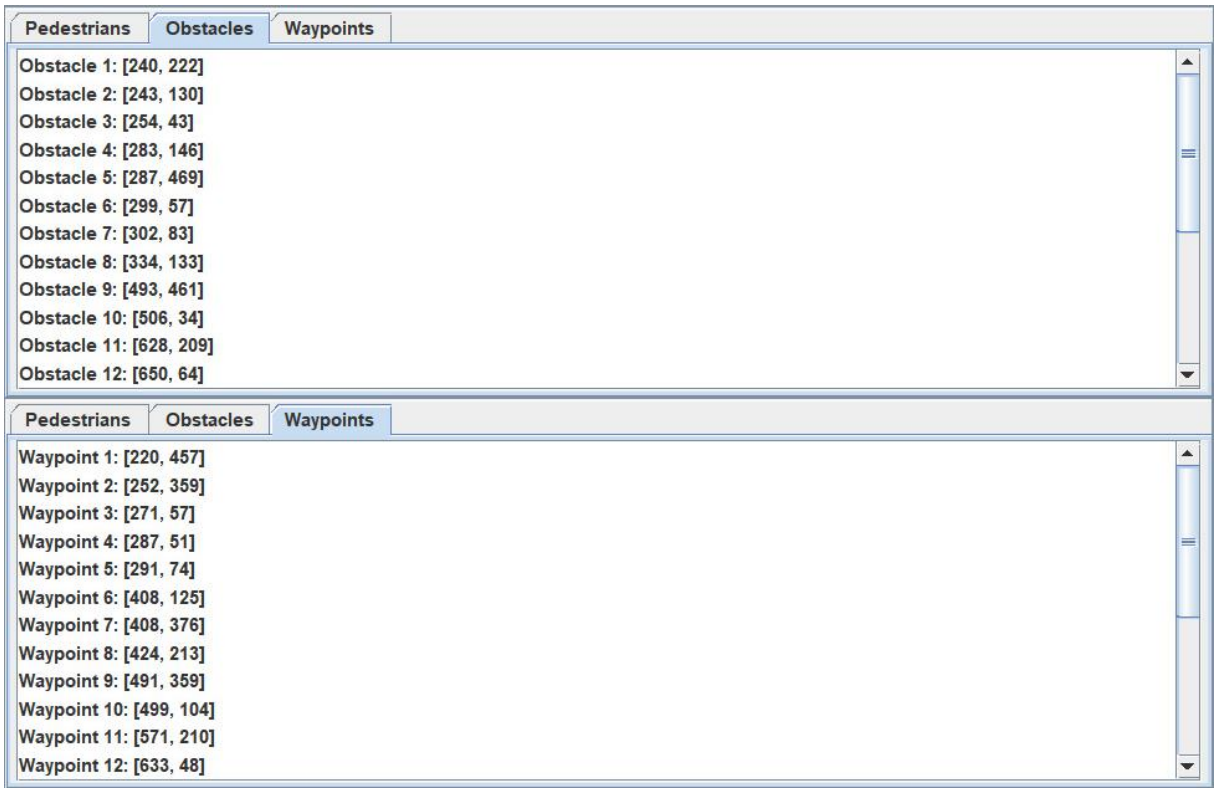

Fig. 2.7: aspetto dei pannelli di ostacoli e obiettivi attivi dopo aver avviato la simulazione

2.4 Conclusioni

In questo capitolo si sono descritti gli aspetti più superficiali del simulatore, si sono esposte le funzionalità delle sue componenti, i ruoli e il comportamento delle entità simulate; tuttavia, il movimento dei pedoni è il risultato di un processo di calcolo complesso, basato su alcune delle tecniche già descritte nel capitolo 1. Nel prossimo capitolo si scenderà nel dettaglio degli algoritmi di *flocking*, di come questi si siano evoluti nel corso del tempo e di come sono stati recuperati, migliorati e implementati in *Crowd Simulator*.

3 GLI ALGORITMI DI *FLOCKING* E DI MOVIMENTO

Nel capitolo 1 si è trattato in maniera ampia ma poco approfondita delle ricerche nel campo della simulazione di folle di pedoni, mentre nel capitolo 2 si è descritto il programma *Crowd Simulator* senza specificarne il funzionamento a basso livello. Questo capitolo si pone l'obiettivo di colmare il vuoto tra queste due prime trattazioni, entrando nel dettaglio dei già citati algoritmi di *flocking*, di come questi sono implementati in *Crowd Simulator* e del modo in cui vengono estesi per costituire l'insieme degli *steering behavior* dei pedoni. Nel paragrafo 3.1 si descrive l'idea originale dai quali ha sviluppo, poi nel paragrafo 3.2 e 3.3 si analizza come questo algoritmo venga esteso e utilizzato in *Crowd Simulator* rispettivamente nella sua prima e seconda versione.

3.1 L'idea originale

Il primo a mostrare uno studio importante in merito agli algoritmi di *flocking* fu il professor Craig Reynolds nel 1987, nel suo articolo intitolato *Flocks, herds and schools: A distributed behavioral model* [2]. L'intento iniziale non era la simulazione di folle di pedoni, bensì cercare un modo per simulare il comportamento di stormi di uccelli; ogni entità nel lavoro di Reynolds prende il nome di *boid*. Ogni *boid* è implementato come un'entità autonoma che naviga in base alla sua percezione locale dell'ambiente circostante, alle leggi della fisica simulata che ne regolano il movimento e a un insieme di comportamenti. Il movimento aggregato dello stormo è il risultato della fitta interazione dei comportamenti relativamente semplici delle singole entità simulate.

Lo studio di Reynolds è fortemente correlato ai già esistenti sistemi di simulazione di particelle [43], questi ultimi sono collezioni di un gran numero di elementi semplici individuali. Le particelle vengono create, invecchiano e muoiono. Durante la loro vita hanno determinati comportamenti che possono alterare lo stato della particella stessa, che consiste in colore, opacità, posizione e velocità. Il lavoro di Reynolds è una generalizzazione dei sistemi di particelle, ma tra una particella e un *boid* ci sono tre differenze significative, il *boid* infatti: è un oggetto geometrico più complesso, non un semplice punto; al contrario della particella ha un orientamento nello spazio; racchiude al suo interno una serie di comportamenti più complessi di quelli descritti in letteratura per le particelle, in particolare, queste non hanno bisogno di tenere in considerazione la presenza delle altre entità vicine, mentre i *boid* devono costantemente monitorare i loro vicini per assumere un comportamento verosimile.

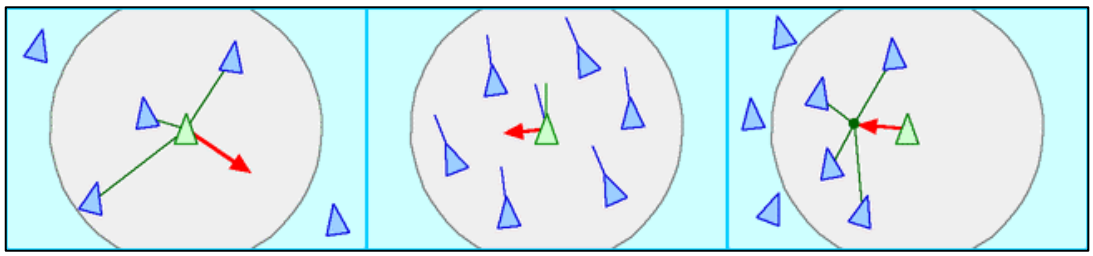

Fig. 3.1: Da sinistra a destra, il vettore rosso rappresenta rispettivamente le forze di separazione, allineamento e coesione dell'entità verde. Il cerchio bianco indica l'area locale del *boid* verde, le entità all'interno di questo cerchio sono considerate ai fini del calcolo delle forze di coesione, allineamento e separazione.

(Fonte: [3])

Nell'articolo, Reynolds parla di un approccio basato su tre regole che governano i movimenti delle singole entità simulate: una forza di separazione che allontana le entità dai propri vicini per evitare casi di affollamento locale e collassamento di tutte le entità in un unico punto; una forza di allineamento, che ha la funzione di regolare la velocità del *boid* rispetto alle entità vicine e allinearlo secondo la direzione e il verso di questi stessi vicini; una forza di coesione che spinge l'entità verso la posizione media (baricentro) dei suoi vicini. Queste forze sono rappresentate in Fig. 3.1: Da sinistra a destra, il vettore rosso rappresenta rispettivamente le forze di separazione, allineamento e coesione dell'entità verde. Il cerchio bianco indica l'area locale del *boid* verde, le entità all'interno di questo cerchio sono considerate ai fini del calcolo delle forze di coesione, allineamento e separazione.

Grazie al solo utilizzo di queste tre forze, lo stormo di entità può spostarsi in un ambiente virtuale semplice. Reynolds nel suo approccio considera anche un ulteriore livello di complessità, discutendo un metodo per la prevenzione delle collisioni con ostacoli presenti nell'ambiente circostante. Un primo metodo prevede l'utilizzo di un campo di forza tramite il quale l'ostacolo in questione respinge il *boid* con una forza inversamente proporzionale alla distanza dell'entità dal suo bordo. Il problema di questo primo approccio è che se un *boid* si avvicina a un ostacolo circondato da un campo di forza con un'angolazione tale da essere esattamente opposta alla direzione del campo di forza, il boide non riesce ad aggirarlo. In questo caso il campo di forza serve solo a rallentare l'entità, respingendola indietro senza fornire alcuna spinta laterale. Il secondo approccio alla prevenzione delle collisioni è detto *steer-to-avoid* ed è più funzionale in quanto si basa sul concetto che ogni entità conosce cosa gli si trova dinanzi. Il *boid* considera solo gli ostacoli che si trovano direttamente di fronte a lui, lavorando nello spazio prospettico locale, trova il bordo dell'ostacolo su cui avverrà l'impatto, viene quindi calcolato un vettore che indirizza l'entità verso un punto oltre il bordo dell'ostacolo.

3.2 *Flocking* in *Crowd Simulator: versione 1*

L'algoritmo di movimento presente nell'attuale versione del simulatore consiste in una leggera rivisitazione dell'algoritmo di *flocking* originale. Il metodo è applicato a una folla eterogenea seguendo un approccio disaccoppiato e *agent-based*, in quanto ogni pedone ha un certo livello di intelligenza e riesce a improvvisare svariati comportamenti a seconda del contesto. Ai fini dell'effettiva implementazione degli algoritmi di Reynolds si è attinto dal lavoro del professor Daniel Shiffman, che nel suo sito [44] propone un'implementazione Java di una versione semplificata del *flocking*. Shiffman sviluppa la simulazione 2D di uno stormo di uccelli in uno spazio virtuale libero di ostacoli. In questo approccio viene utilizzata la classe Java *PVector* per rappresentare le forze fisiche in gioco, ogni istanza di tale classe è un vettore fisico bidimensionale, con modulo, direzione e verso.

Anche nel programma *Crowd Simulator* si utilizza la classe *PVector* e, per riprodurre il comportamento della folla, si utilizzano i concetti già citati di separazione e coesione. In questo approccio non si utilizza il vettore di allineamento in quanto, sebbene questa forza sia necessaria ai fini della fluidità dei movimenti nella rappresentazione delle virate di uno stormo, non lo è altrettanto nel cambio di direzione di un pedone, per questo motivo è stato considerato un parametro superfluo. Le funzioni per il calcolo di coesione e separazione sono state prese dal sito del professor Shiffman [44], in seguito sono state opportunamente modificate e adattate come mostrato in Appendice A, rispettivamente in a. 1 e in a. 2. In aggiunta alle forze appena citate, per calcolare il movimento di ogni pedone si utilizzano anche due vettori aggiuntivi: uno che determini lo spostamento del pedone per scansare un ostacolo statico, così come mostrato in a. 3; uno che indirizza il pedone verso il suo obiettivo corrente, viene calcolato come mostrato

in a.4. Le forze appena descritte vengono unite all'interno di un'unica forza risultate, determinando il movimento del pedone. Durante questa fase l'influenza e il peso di ciascuna delle forze dipendono dal contesto in cui il pedone si trova e avviene tramite due tecniche: *switching* e *blending*. Nello *switching*, man mano che le circostanze della camminata cambiano nell'ambiente simulato, il pedone può "passare" da una modalità di comportamento all'altra, come mostrato in a. 5: algoritmo di *switching/blending* delle forze applicate al pedone. Invece, nel *blending* questi comportamenti, che agiscono in parallelo, sono "mescolati" insieme in un'unica forza risultante. Ad esempio, in situazioni normali, i pedoni che si muovono nell'ambiente simulato verso la loro destinazione fondono i comportamenti di *flocking* e con quelli di orientamento verso l'obiettivo in un unico vettore direzionale, per consentire al gruppo di camminare verso la meta e rimanere coesi allo stesso tempo. Inoltre, supponiamo che un gruppo di pedoni si muova in un determinato ambiente e si avvicinino a un ostacolo o ad altri pedoni dello stesso gruppo o di gruppi diversi. Questa situazione porta a passare da una fase in cui la priorità è il movimento verso l'ostacolo, a una in cui la priorità è l'evitamento delle collisioni, anche a costo di spezzare la coesione del gruppo.

Per quanto riguarda il controllo delle collisioni con l'edificio, i pedoni non hanno una forza che li spinge altrove dai muri della struttura, poiché l'unico motivo per il quale dovrebbero avvicinarsi a un muro è perché sono diretti verso una porta. In questo algoritmo esiste solamente un metodo che impedisce ai pedoni di attraversare i muri, quest'ultimo è implementato attraverso un sistema di BNM (Boundary Node Method), che associa ad ogni pedone una *safety-zone*, la quale costituisce il suo spazio personale, formata come descritto in figura Fig. 3.2: *safety-zone* del pedone e bordi che ne delimitano lo spazio di collisione

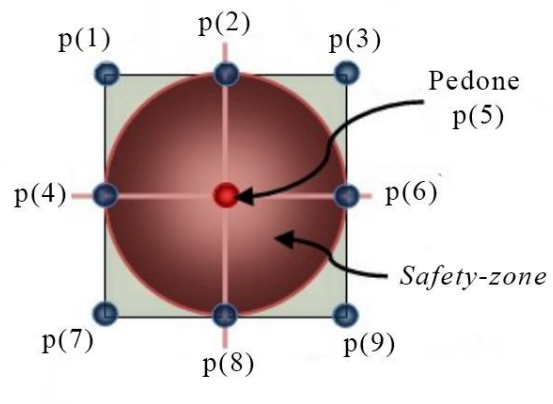

Fig. 3.2: *safety-zone* del pedone e bordi che ne delimitano lo spazio di collisione

(Fonte: [45])

Ogni pedone si trova al centro di un sistema di punti attraverso i quali si rilevano le collisioni con l'edificio. A seconda di quale collide, al pedone viene vietato di procedere in una determinata direzione: se collide p(2) il pedone non potrà procedere verso l'alto, se collide p(4) non potrà procedere verso sinistra e così via. Per evitare problemi quando un pedone cammina nei pressi delle porte dell'edificio, si effettuano anche controlli sui punti che costituiscono gli angoli della figura.

3.2.1 L'algoritmo di movimento completo

Questa versione dell'algoritmo di movimento produce in output la posizione che il singolo pedone dovrà assumere dopo aver compiuto un passo. Il calcolo avviene in maniera ciclica e continua, a ogni iterazione il passo successivo del pedone è frutto di un processo che può essere riassunto in cinque fasi:

Fase 1: Esecuzione della funzione *flock():*

a) la funzione calcola i vettori di separazione, coesione, direzione verso la quale il pedone deve dirigersi per giungere al suo obiettivo e prevenzione delle collisioni con ostacoli statici, per poi assegnare un certo peso a ciascuno dei vettori, ovvero l'influenza che ognuno di loro deve avere nel calcolo della posizione successiva;[8]

b) unione dei vettori appena calcolati, il risultato di questa addizione costituirà il vettore accelerazione del pedone.

Fase 2: Esecuzione della funzione *update()*:

a) somma il vettore accelerazione al vettore velocità, che viene poi ridimensionato in base alla velocità massima del pedone;

b) A questo punto il vettore velocità sommato al vettore posizione determina la prossima posizione del pedone; dunque, si utilizza questa conoscenza per controllare se nel prossimo passo si verificherà una collisione con un muro dell'edificio. Se non si verificherà nessuna collisione allora la velocità viene sommata alla posizione e si procede con l'algoritmo, altrimenti si vieta al pedone di proseguire nella direzione nella quale è stata rilevata la collisione.

Fase 3: Esecuzione della funzione *updateBounds()* che cura gli aspetti della parte grafica relativa al pedone;

Fase 4: Infine, si eseguono i controlli sull'eventuale raggiungimento di un obiettivo da parte del gruppo:

a) Se il gruppo ha raggiunto una delle porte dell'edificio allora non avviene alcuna sosta.

b) Se il gruppo ha raggiunto un obiettivo in cui può riposare allora vi sosterà fino a quando ogni pedone del gruppo non ha nuovamente raggiunto il massimo delle sue forze.

c) Se il gruppo ha raggiunto un generico obiettivo, allora parte un timer di durata compresa in un certo intervallo di tempo, allo scadere del quale la visita finisce.

Fase 5: Si verifica quale sia il prossimo obiettivo e lo si imposta come meta corrente. In questa fase vengono considerati obiettivi anche le porte dell'edificio, chiaramente presso queste non verrà effettuata nessuna sosta.

[8] Tutti gli algoritmi citati in questo punto sono discussi nel dettaglio in Appendice A

3.2.2 Criticità e soluzioni

Questa versione dell'algoritmo sebbene sia funzionante presenta comunque delle criticità, la più evidente è la gestione delle collisioni con gli ostacoli statici. Quest'ultima è implementata attraverso un sistema che emula un campo di forza attorno agli ostacoli; perciò, è possibile che si verifichi quanto già descritto nel paragrafo 3.1, ossia che in determinate situazioni un pedone sembri rimbalzare su un ostacolo piuttosto che aggirarlo, risultando nell'assunzione di un atteggiamento poco verosimile. Una soluzione semplice a questa dinamica potrebbe essere l'applicazione di un approccio proposto dallo stesso Reynolds, che prevede di dotare ogni pedone di una certa visione centrale che gli permetta di rilevare cosa gli si trova davanti, in modo che agisca di conseguenza. Questa soluzione può essere implementata tramite l'uso di vettori che possono avere configurazioni diverse, come mostrato in Fig. 3.3: Configurazione dei vettori che costituiscono la visione centrale di un pedone simulato Non esistono regole ferree per stabilire quale configurazione sia migliore, ognuna ha le sue particolari caratteristiche. Un vettore singolo con due raggi corti sui lati in questo caso sarebbe probabilmente la scelta migliore, poiché solitamente negli ambienti di simulazione di pedoni non ci sono passaggi stretti, ambienti in cui questa configurazione potrebbe diventare problematica. Un'altra possibile soluzione è utilizzare il già citato BNM per implementare un sistema che permetta di aggirare l'ostacolo sfruttando i punti che circoscrivono lo spazio personale del pedone, secondo un principio simile a quello che governa le collisioni con le pareti dell'edificio.

Un secondo problema dell'algoritmo consiste nel fatto che il movimento dei pedoni non appare sempre verosimile. Capita talvolta che i pedoni sembrino fluttuare piuttosto che camminare, questo problema è dato dalla gestione dell'equilibrio delle forze che descrivono il movimento del pedone (equilibrio esaminato in a.5), che risulta non essere sempre corretto. Una soluzione a tale problema potrebbe essere la semplice ridistribuzione di forze in alcuni contesti, anche se potrebbe non essere la più efficiente. Nella seconda versione dell'algoritmo descritta nel paragrafo 243.3 si risolve questo problema introducendo un *path planner* per la gestione del percorso del pedone. Questo aspetto, che verrà approfondito nel paragrafo successivo, fa in modo che ogni pedone utilizzi le conoscenze globali in merito all'ambiente virtuale per sapere, in linea di massima, quale percorso andrà a compiere durante il corso della simulazione, improvvisando un comportamento solamente in caso di necessità. Al contrario, in questa prima versione dell'algoritmo ogni pedone improvvisa il suo comportamento a ogni istante della simulazione, causando talvolta dei comportamenti poco verosimili.

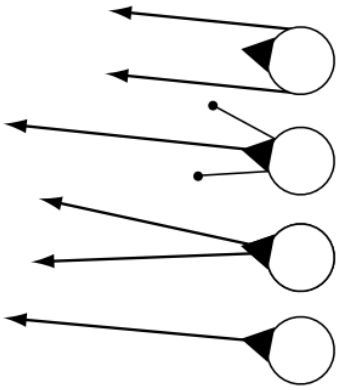

Fig. 3.3: Configurazione dei vettori che costituiscono la visione centrale di un pedone simulato.

(Fonte: [46])

Un terzo problema della prima versione dell'algoritmo è che non è in grado di generalizzare su ogni tipo di ambiente di simulazione. Quest'ultimo è stato creato in maniera tale da funzionare bene in un ambiente in cui ci siano delle stanze e un corridoio che le collega; tuttavia, se si utilizzasse all'interno di un labirinto non funzionerebbe allo stesso modo. Questo problema è dovuto alla gestione degli obiettivi e ai metodi utilizzati per guidare i pedoni verso il loro raggiungimento. Un buon modo per risolvere questo problema consiste nell'introduzione di un metodo di *path finding* all'interno del calcolo del percorso, che sia quindi in grado di calcolare un cammino a prescindere dalla struttura dell'ambiente di simulazione.

3.3 *Flocking* in Crowd Simulator: versione 2

Il motivo principale per cui nasce Crowd Simulator è la raccolta di informazioni ricavate da un algoritmo ben più innovativo di quello descritto nel paragrafo 203.2. Quest'ultimo è un metodo per la simulazione di folle di pedoni eterogenee con gestione disaccoppiata del movimento [45]. Le entità su cui lavora sono le stesse già descritte nel paragrafo 2.1, cambia solamente l'approccio con il quale viene gestito lo spostamento dei pedoni.

Il metodo proposto utilizza un modello microscopico multi-gruppo per generare una traiettoria in tempo reale per ogni pedone durante la navigazione nell'area dell'edificio. Inoltre, viene introdotto un modello *agent-based* per modellare i comportamenti dei singoli pedoni. Grazie a questo duplice approccio, i pedoni utilizzano la conoscenza locale per evitare le collisioni e interagire con gli altri membri della folla, mentre utilizzano la conoscenza globale per la pianificazione a lungo termine e per essere in grado di orientarsi verso il loro obiettivo corrente. L'insieme degli *steering behavior* che gestiscono il movimento racchiudono le forze di *flocking*, ossia separazione, coesione e allineamento, così come discusse nel paragrafo 3.1 e una forza atta a indirizzare il pedone verso il suo obiettivo. Queste forze, così come nella prima versione dell'algoritmo, vengono fuse insieme e l'importanza di ciascuna di esse varia a seconda del contesto, secondo i concetti di *switching* e *blending* già descritti nel paragrafo 3.2.

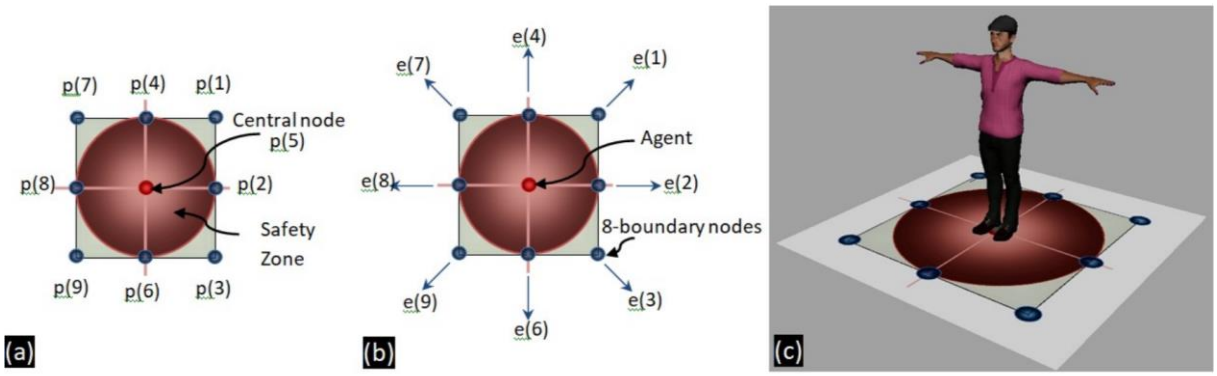

Fig. 3.4: un elemento quadrilaterale con 9 nodi che rappresenta la *safety-zone* del pedone (a), ognuno di questi punti corrisponde a una direzione di moto (b). (c) mostra come questo elemento si applica a un pedone.

(Fonte: [45])

Per quanto riguarda la prevenzione delle collisioni con gli ostacoli statici, si implementa una tecnica diversa da quella discussa per la prima versione dall'algoritmo. Invece che simulare la

presenza di un campo di forza attorno all’ostacolo si implementa un sistema basato su un BNM esteso, che verrà usato per generare un percorso senza collisioni per ogni pedone quando si avvicina agli ostacoli in tempo reale. L'utilizzo della pianificazione globale del percorso in contesti real-time è difficile da modellare utilizzando un modello *agent-based*, perché diventa computazionalmente costoso. Pertanto, i modelli basati su agenti utilizzati in questo algortimo separano la prevenzione delle collisioni locali dalla pianificazione globale del percorso. Sulla base del metodo BNM esteso, ogni pedone nella folla è simulato da un elemento quadrilatero a nove nodi, come descritto in Fig. 3.4: un elemento quadrilaterale con 9 nodi che rappresenta la *safety-zone* del pedone (a), ognuno di questi punti corrisponde a una direzione di moto (b). (c) mostra come questo elemento si applica a un pedone..

3.4 Conclusioni

Nel corrente capitolo sono stati descritti, nei limiti di quanto sia attualmente possibile affermare, gli algoritmi di movimento sui quali ci si è concentrati durante la codifica di *Crowd Simulator*. Come è già stato detto, non è possibile eseguire una trattazione esaustiva della seconda versione dell’algoritmo di movimento. Tuttavia, si possono comunque trarre delle conclusioni in merito agli utilizzi futuri del simulatore e della sua collocazione nel panorama della ricerca scientifica.

CONCLUSIONI E SVILUPPI FUTURI

Allo stato attuale il programma *Crowd Simulator* è un programma incompleto, ma con diverse proposte di miglioramento dell'interfaccia. Ai fini del completamento del software è fondamentale l'aggiunta della seconda versione dell'algoritmo di movimento dei pedoni, così da poter collezionare dati in merito al suo funzionamento. Tra gli altri sviluppi futuri del software si prevede la possibilità di poter automatizzare l'esecuzione delle simulazioni, la quale diverrebbe ciclica e continua; questo miglioramento cancellerebbe uno dei limiti del programma, ossia il fatto di poter eseguire una sola simulazione alla volta, che deve essere avviata manualmente dall'utente. Oltre all'esecuzione automatica, potrebbe anche essere implementato un sistema di *thread* tramite il quale eseguire simultaneamente più simulazioni, in modo da raccogliere un numero di dati maggiore.

Tra il 2020 e il 2022 la ricerca scientifica si è concentrata particolarmente su modelli di evacuazione di luoghi chiusi in situazioni di emergenza, comportamenti di folle, gruppi e individui e diverse metodologie di navigazione. Le tecniche che sembrano essere maggiormente utilizzate al giorno d'oggi (e probabilmente nei prossimi anni) sono: quelle data-driven, l'uso di algoritmi di machine learning e metodi che consentono ai singoli pedoni di avere una propria personalità e un certo livello di intelligenza. Nel futuro della ricerca ci si aspetta una tendenza verso il miglioramento degli aspetti macroscopici e microscopici dei comportamenti di un insieme di pedoni più o meno denso, così da produrre simulazioni che permettano di fare uno zoom sulla folla per osservarne i comportamenti a diversi livelli di dettaglio [18]. Inserito in questo panorama della ricerca scientifica, una volta effettuati i miglioramenti sopra citati, *Crowd Simulator* potrebbe essere un ottimo strumento di raccolta di dati, le sue applicazioni potrebbero favorire lo sviluppo di algoritmi innovativi che consentirebbero l'avanzamento degli studi in questo ambito.

APPENDICE A

La presente appendice ha lo scopo di fornire i dettagli tecnici di implementazione del simulatore, nel paragrafo A.1 si descrive brevemente l'hardware e il software con il quale è stato svolto l'intero progetto di codifica, mentre i paragrafi successivi forniscono una spiegazione dettagliata del funzionamento degli algoritmi di movimento già descritti durante la trattazione. In particolare, nel paragrafo A.2 ci si concentra unicamente sulla prima versione dell'algoritmo di movimento e si analizzano le quattro forze necessarie allo spostamento dei pedoni, trascurando la struttura che le unisce (già descritta in maniera sommaria nel paragrafo 3.2.13.2.1). Nel paragrafo A.3, al contrario, si analizza solamente la struttura che mette insieme le varie componenti degli *steering behavior* descritti in 3.3 per la seconda versione dell'algoritmo di movimento. Non si scendere nei dettagli di questo secondo algoritmo in quanto, come già detto, in fase di scrittura di questo elaborato si trova ancora in fase di miglioramento.

A.1 Hardware e software nella codifica di *Crowd Simulator*

Lo sviluppo del programma *Crowd Simulator* è avvenuto interamente su VivoBook ASUS Laptop X530FN S530FN con sistema operativo Windows 11. Si è utilizzato il framework *Swing* in associazione con Java 18 e compilatore Javac. Per l'utilizzo della classe PVector è stato necessario importare il package esterno *processing.core*, mentre per l'utilizzo della classe GSon, necessaria ai fini della produzione dei file in sintassi JSON, è stato importato il package esterno *com.google.gson*. Il numero di righe di codice prodotte, comprensive di commenti, è pari a 3800, il tempo impiegato nella costruzione del sistema è stato di circa 400 ore.

A.2 Movimento dei pedoni: versione 1

In questo paragrafo sono riportate in pseudo-codice e descritte le funzioni che mostrano il calcolo dei quattro vettori attorno ai quali si costruisce l'intero sistema che gestisce il movimento dei pedoni nell'attuale versione di *Crowd Simulator*. La prima è la forza di coesione, calcolata come espresso in a.1.

Nel calcolo della coesione si definisce innanzitutto un vettore *sum* (riga 1, algoritmo a.1) che raccoglie la somma delle posizioni dei membri del gruppo, così da poterne fare una media e ottenere il baricentro del gruppo. Se *p* è l'unico membro del suo gruppo allora non esiste un baricentro, quindi si restituisce in output un vettore nullo (riga 16, algoritmo a.1). Se il gruppo è composto da più membri, per ognuno di questi si valuta quale sia la sua distanza rispetto a *p* (riga 5, algoritmo a.1) e si considera tale valore nel calcolo del baricentro del gruppo (riga 7, algoritmo a.1). Una volta ottenuta la posizione di tutti i membri del gruppo, il vettore *sum* viene diviso per il numero di membri effettivi, così da trovare la posizione media, e viene restituito un vettore che indirizza *p* verso tale posizione (riga 12, algoritmo a.1). La funzione *direction* (riga 12, algoritmo a.1) è la medesima che viene utilizzata anche per fare in modo che i pedoni si dirigano verso gli obiettivi presenti nell'ambiente virtuale (algoritmo a.4). La funzione di coesione implementata in questo algoritmo differisce dall'idea originale di Reynolds (quindi anche dall'implementazione del professor Shiffman) in quanto la folla è organizzata in gruppi. Reynolds nel suo studio considerava un unico grande gruppo, quindi aveva la necessità di definire uno spazio attorno a ogni entità all'interno del quale ognuna di queste potesse percepire la presenza di vicini a cui affidarsi per il calcolo della coesione, così come mostrato in Fig. 3.1.

In *Crowd Simulator* i pedoni dello stesso gruppo devono rimanere coesi a prescindere dalla loro vicinanza; dunque, la forza di coesione viene calcolata prendendo in considerazione tutti i membri del gruppo, anche se questi risultano lontani tra loro.

```
Input: membri del gruppo a cui appartiene p
Output: vettore coesione
```

```
1  sum ← new(0,0)
2
3  if groupHasOtherMembers then
4      for each i : memberOfTheGroup do
5          dist ← distance(positionₚ, position[i])
6          if dist > 0 then
7              sum ← sum + position[i]
8          end if
9      end for
10
11     sum ← sum / numberOfMembers
12     return direction(sum)
13 end if
14
15 else
16     return sum
```

a. 1: algoritmo di calcolo del vettore coesione

La seconda forza in gioco è quella della separazione, calcolata nell'algoritmo a.2. Il vettore separazione viene calcolato in maniera simile a come viene descritto nello studio originale di Reynolds. Dato un pedone *p*, la funzione prende in input l'intera folla di pedoni e per ognuno di questi se ne calcola la distanza rispetto a *p* (riga 6, algoritmo a.2) e, se questo valore è minore di un certo *desiredSeparation* (riga 8, algoritmo a.2), si calcola il vettore che consente di mantenere le distanze, lo si normalizza, si scala il suo modulo sulla base della distanza di *p* dal pedone *i* e lo si somma al vettore *steer* (riga 9-12, algoritmo a.2), il quale consente di evitare tutte le possibili collisioni nel caso in cui ci siano da considerare più pedoni. A ogni ciclo, se ci sono pedoni da evitare, si aggiorna il *counter* (riga 13, algoritmo a.2) che tiene traccia di quanti pedoni sono troppo vicini a *p*. Se il modulo del vettore *steer* è maggiore di zero (riga 17, algoritmo a.2) significa che ci sono dei pedoni da evitare, quindi si pesa il vettore *steer* sulla base del numero di pedoni da evitare (riga 18, algoritmo a.2), per poi normalizzarlo e regolarne la velocità (riga 19-22, algoritmo a.2), infine se ne limita il modulo a una costante *maxForce* (riga 23, algoritmo a.2) per evitare sterzate brusche che sovrastino le altre forze in gioco.

Per quanto concerne la prevenzione delle collisioni con ostacoli statici, questa avviene secondo un approccio molto simile a quello già utilizzato per evitare le collisioni con gli altri pedoni. Nel calcolo di questa forza vengono inizializzati il vettore *steer* (riga 1, algoritmo a.3) e la distanza massima *desiredDistance* oltre la quale il pedone deve iniziare la sua manovra di evitamento dell'ostacolo (riga 2, algoritmo a.3), quest'ultima dipende dalla dimensione del pedone e da quella dell'ostacolo in questione. Per ognuno degli ostacoli presenti nell'ambiente virtuale (riga 4, algoritmo a.3) si verifica se il pedone *p* è troppo vicino a uno di essi (riga 5, algoritmo a.3), in caso affermativo si calcola il vettore che permette di aggirare l'ostacolo così

come già fatto anche in a.2 (riga 6-9, algoritmo a.3), altrimenti la funzione restituisce un vettore nullo (riga 13, algoritmo a.3).

```
Input: pedoni dell'intera folla
Output: vettore separazione
```

```
1  desiredSeparation ← pedestrianWidth
2  counter ← 0
3  steer ← new(0,0)
4
5  for each i : memberOfTheCrowd do
6      dist ← distance(position_p, position[i])
7
8      if dist > 0 AND dist < desiredSeparation then
9          difference ← position_p – position[i]
10         difference.normalize()
11         difference ← difference / dist
12         steer ← steer + difference
13         counter ← counter + 1
14     end if
15 end for
16
17 if steer.module > 0 then
18     steer ← steer / counter
19     steer.normalize()
20     steer ← steer * maxVelocity_p
21     steer ← steer – velocity
22     steer.limit(maxForce)
23 end if
24
25 return steer
```

a. 2: algoritmo di calcolo del vettore separazione

```
Input: elenco degli ostacoli presenti nell'ambiente virtuale
Output: vettore che consente di evitare una collisione
```

```
1  steer ← new(0,0);
2  desiredDistance ← pedestrianWidth + obstacleWidth
3
4  for each i : obstacles do
5      if distance(position_p, position[i]) < desiredDistance then
6          difference ← position_p - position[i]
7          difference.normalize()
8          difference ← difference / distance(position_p - position[i])
9          steer ← steer + difference
10     end if
11 end for
12
13 return steer
```

a. 3: algoritmo di calcolo del vettore di prevenzione delle collisioni con ostacoli statici

Infine, a.4 descrive la funzione che calcola il vettore che dirige un pedone *p* verso un dato obiettivo. La funzione prende in input l'obiettivo verso il quale il pedone ha necessità di dirigersi, quindi viene calcolato il vettore *desired* che va dalla posizione di *p* verso tale obiettivo (riga 1, algoritmo a.4). Quest'ultimo viene ridimensionato sulla base della massima velocità a cui si può spostare il pedone (righe 2-3, algoritmo a.4) e poi viene utilizzato per calcolare l'effettiva direzione verso la quale il pedone deve spostarsi per raggiungere il suo obiettivo (riga 4-5, algoritmo a.4).

Le quattro forze di coesione, separazione, evitamento e direzione devono coesistere allo stesso istante ma con diverse intensità, o si otterrebbe un moto nel quale questi vettori andrebbero costantemente a interferire con la funzione degli altri. Si ha quindi la necessità di trovare un metodo che dosi queste forze, anche dipendentemente dal contesto in cui il pedone si trova. Questo approccio viene descritto nell'algoritmo a.5.

```
Input: posizione dell'obiettivo da raggiungere
Output: vettore con direzione e verso orientati verso l'obiettivo
```

```
1  desired ← posizione_obiettivo - posizione_p
2  desired.normalize()
3  desired ← desired * maxVelocity_p
4  steer ← desired - velocity_p
5  steer.limit(maxForce)
6
7  return steer
```

a. 4: algoritmo di calcolo del vettore che orienta il pedone verso un dato obiettivo

Durante la maggior parte della simulazione le quattro forze già descritte in a.1, a.2, a.3 e a.4 hanno delle diverse priorità nel calcolo del movimento del pedone: la forza di coesione viene moltiplicata per un fattore 1 (riga 1, algoritmo a.5); la forza di separazione viene moltiplicata per 2.2 (riga 2, algoritmo a.5); la forza che spinge il pedone verso l'obiettivo corrente viene moltiplicata per 1.2 (riga 3, algoritmo a.5); la forza che consente al pedone di evitare la collisione con gli ostacoli statici viene moltiplicata per 0,6. Questi valori appena elencati potrebbero rimanere statici in assenza di un edificio con il quale interagire, tuttavia, la sua presenza pone i presupposti per creare situazioni all'interno del quale questo equilibrio tra le forze va modificato. Quando un pedone sta camminando lungo il muro di un edificio (riga 6, algoritmo a.5), la coesione viene ridotta a zero (riga 7, algoritmo a.5), poiché potrebbero esserci dei membri del suo gruppo che sono già dall'altra parte di quel muro,[9] quindi la priorità viene data al fatto di aggirare il muro piuttosto che stare unito al gruppo. Il secondo caso in cui il contesto rende necessario la modifica del peso delle forze è quando i pedoni di un gruppo si trovano nell'area esterna dell'edificio (riga 11, algoritmo a.5), in questo caso si aumenta la coesione per farli riunire il prima possibile, ma allo stesso tempo si incrementano anche le forze di separazione e direzione (righe 12-14, algoritmo a.5), in modo da non far prevalere quella di coesione, che farebbe collassare il gruppo verso un unico punto. Il terzo caso si verifica quando un pedone si allontana dal resto del gruppo (riga 17, algoritmo a.5), in questo caso si favorisce la coesione, ma limitando leggermente la forza del vettore direzione (riga 18-20, algoritmo a.5).

[9] Se un pedone cammina rasente a un muro della struttura molto probabilmente si sta dirigendo verso una delle porte dell'edificio.

```
Input: vettori risultanti dai calcoli di a.1, a.2, a.3, a.4
Output: //
```

```
1  cohMultiplicator ← 1
2  sepMultiplicator ← 2.2
3  dirMultiplicator ← 1.2
4  avoidMultiplicator ← 0.6
5
6  if p.collidedWithWall then
7      cohMultiplicator ← 0
8      avoidMultiplicator ← 0
9  end if
10
11 else if p.isOutOfTheBuilding then
12     cohMultiplicator ← 1.5
13     sepMultiplicator ← 2.5
14     dirMultiplicator ← 1.3
15 end if
16
17 else if p.isFarFromTheGroup then
18     cohMultiplicator ← 1.5
19     sepMultiplicator ← 2.5
20     dirMultiplicator ← 1.1
21 end if
22
23 cohesion ← cohesion * cohMultiplicator
24 separation ← separation * sepMultiplicator
25 direction ← direction * dirMultiplicator
26 avoid ← avoid * avoidMultiplicator
```

a. 5: algoritmo di *switching/blending* delle forze applicate al pedone

A.3 Movimento dei pedoni: versione 2

A ogni fotogramma chiave (*keyframe*) i pedoni si muovono in avanti rispetto alla loro posizione corrente alla nuova posizione aggiornata ($Position_{pe}$) con la velocità di camminata $Velocity_{pe}$. L'algoritmo a.6 spiega il movimento dei pedoni nell'ambiente virtuale dal punto di partenza al punto di arrivo e il processo è illustrato nelle seguenti fasi.

Fase 1: L'algoritmo prende in input i dati inerenti all'ambiente di simulazione e alle sue entità: il numero di frames che andranno a comporre l'intera animazione *nFrames*; il numero di gruppi di pedoni *nGroups*; la lista di attivazione dei gruppi $Activate_{groups}$; la lista di terminazione dei gruppi $Terminate_{groups}$; la lista di pedoni nella folla $Pedestrians_{crowd}$; il numero di pedoni nella folla $nPedestrians_{crowd}$; la lista di obiettivi per ogni gruppo di pedoni $Goals_{crowd}$; la lista delle posizioni di ogni membro della folla $Positions_{crowd}$; la costante che stabilisce di quanto deve diminuire l'energia dei pedoni ad ogni keyframe c_{s1}.

Fase 2: Ad ogni *keyframe* (riga 1, algoritmo a.6), il valore di attivazione di ogni gruppo $Activate_{Groups}[g_r]$ è impostato su "on" quando keyframe raggiunge il valore $kFrame_{activate}[g_r]$ (riga 2-3, algoritmo a.6). Questo valore consente ai membri, inizialmente fermi, di iniziare a camminare, dunque, per ogni gruppo attivo si definisce:

a) l'indice dell'obiettivo $Goals_{crowd}[g_r]$ (riga 7, algoritmo a.6);

b) i pedoni nel gruppo $Pedestrians_{crowd}[g_r]$ (riga 9, algoritmo a.6);

c) numero di pedoni nel gruppo n$Pedestrians_{crowd}[g_r]$ (riga 10, algoritmo a.6);

d) gli obiettivi $Goals_{crowd}[g_r]$ per il gruppo n$Pedestrians_{crowd}[g_r]$ (riga 11, algoritmo a.6).

Fase 3: Per ogni pedone (riga 12, algoritmo a.6) la sua posizione corrente $Position_{pe}$ è ottenuta in tempo reale dalla scena, mentre la sua velocità $Velocity_{pe}$ e la sua energia $Energy_{pe}$ sono ottenute dagli attributi del pedone contenuto in $Pedestrians_{group}[p_e]$ (riga 14, algoritmo a.6).

Fase 4: Ad ogni keyframe viene anche calcolata la nuova $Velocity_{pe}$ per ogni pedone in $Pedestrians_{group}[p_e]$ nei gruppi attivi. La velocità è limitata ad un certo valore che viene influenzato da c_{s1}. Allo stesso tempo viene anche calcolata la nuova $Position_{pe}$ sulla base degli *steering behavior* (riga 26, algoritmo a.6).

Fase 5: Si verifica che non avvengano collisioni con altri pedoni (riga 27, algoritmo a.6), con i quali deve essere mantenuta sempre una determinata distanza minima, in caso di avvenuta collisione si procede subito con il calcolo della nuova posizione del pedone (riga 28, algoritmo a.6). Si verifica anche se avvengono collisioni con ostacoli statici (riga 30, algoritmo a.6), in caso affermativo si utilizza il metodo BNM per calcolare una nuova $Position_{pe}$ per il pedone interessato.

Fase 6: Ad ogni keyframe il livello di energia di ogni pedone viene decrementato (riga 15, algoritmo a.6). Quando il valore di $Position_{pe}$ scende sotto a un certo valore soglia tutti i pedoni del gruppo si dirigeranno verso un punto di ristoro dopo aver concluso il compito che stavano svolgendo (riga 33-34, algoritmo a.6). Il gruppo rimane presso il punto di ristoro fino a quando ogni componente ha di nuovo raggiunto il livello massimo di energia. A questo punto continueranno a dirigersi verso i loro obiettivi (riga 36-41, algoritmo a.6).

Fase 7: Ad ogni keyframe, se la distanza tra un pedone del gruppo $Pedestrians_{group}$ situato in $Position_{pe}$ e un obiettivo $Goals_{group}[Goal_i]$ è minore di c_{s2}, significa che quel gruppo è arrivato nei pressi dell'obiettivo corrente. Dunque, questo gruppo deve cercare un nuovo obiettivo dalla sua lista e aggiornare l'$index(Goal_{index})$ degli obiettivi (riga 18-25, algoritmo a.6).

Fase 8: Ad ogni keyframe i pedoni cambiano la loro posizione attuale in $Position_{pe}$ in tempo reale, subito dopo, vengono aggiornati anche gli attributi del pedone, includendo $Velocity_{pe}$ ed $Energy_{pe}$ in $Pedestrians_{group}[p_e]$ (riga 42-44, algoritmo a.6).

Fase 9: Quando il gruppo $Pedestrians_{group}[g_r]$ raggiunge l'ultimo obiettivo $Goals_{group}[last]$, il valore di $Terminate_{groups}[g_r]$ viene impostato su "on" e il valore di $Activate_{groups}[g_r]$ viene impostato su "off" (riga 18-25, algoritmo a.6).

```
Input: nFrames, nGroups, Activate_groups, Terminate_groups, Pedestrians_crowd,
nPedestrians_crowd, Goals_crowd, Positions_crowd, c_s1
1  for keyframe ← 1 to nFrames do
2      if keyframe = kFrame_activate then
3          Activate_groups[g_r] ← "on"
4      end if
5      pc ← 0, Goals_index([1] * nGroups)
6      for g_r ← 1 to nGroups do
7          pc ← pc + 1, Goal_i ← Goals_index[g_r]
8          if Terminate_groups[g_r] = "off" AND Activate_groups[g_r] ← "on" then
9              Pedestrians_group ← Pedestrians_crowd [g_r]
10             nPedestrians_group ← nPedestrians_crowd[g_r]
11             Goals_group ← Goals_crowd[g_r]
12             for p_e ← 1 to nPedestrians_group do
13                 Compute Separation, Cohesion, Allignment
14                 (Position_pe, Velocity_pe, Energy_pe) ← Pedestrians_group[p_e]
15                 Energy_pe ← Energy_pe – c_s1
16                 Pedestrians_group[p_e] ← Energy_pe
17                 if Energy_pe > minimumEnergy_pe then
18                     if dist(Position_pe, Goals_group[Goal_i]) < c_s2 then
19                         Goal_i ← Goal_i + 1
20                         updateGoals_index[g_r] ← Goal_i
21                         if Goal_i = len(Goals_group) then
22                             Terminate_groups[g_r] ← "on"
23                             Activate_groups[g_r] ← "off"
24                         end if
25                     end if
26                     Compute new Velocity_pe and Position_pe
27                     if indCollision = "true" then
28                         Position_pe ← pedestrian collision avoidance
29                     end if
30                     if obstacleCollision = "true" then Position_pe ← BNM
31                     end if
32                 else
33                     stay in service point, modify Goals_group and Energy_pe
34                     update Pedestrians_group[p_e] with new Energy_pe
35                     if staying time is finished then
36                         for p_e ← 1 to nPedestrians_group do
37                             new Energy_pe ← maximum Energy_pe
38                             update Pedestrians_group[p_e] ← new Energy_pe
39                         end for
40                     end if
41                 end if
42                 Position_crowd[p_c] ← Position_pe
43                 update keyframe and Position_pe
44                 update Pedestrians_group[p_e] ← new(Position_pe, Velocity_pe)
45             end for
46         end if
47     end for
48 end for
```

a. 6: gestione del movimento nella seconda versione dell'algoritmo

BIBLIOGRAFIA

[1] N. Wiener, Cybernetics, or Control and communication in the Animal and the Machine, Parigi: Hermann Editeurs, 1948.

[2] C. W. Reynolds, «Flocks, herds and schools: A distributed behavioral model,» *ACM SIGGRAPH Computer Graphics Volume 21, Issue 4*, pp. 25-34, 1987.

[3] C. Reynolds, «Steering Behaviors For Autonomous Characters,» in *Game Developers Conference*, San Jose, California, 1999.

[4] R. C. Arkin, «Motor schema based navigation for a mobile robot: An approach to programming by behavior,» *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4, pp. 264-271, 1987.

[5] R. Arkin, «Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by behavior,» *Proceedings of IEEE Conference on Robotics and Automation*, pp. 264-271, 1987.

[6] R. Arkin, «Motor Schema-Based Mobile Robot Navigation,» *International Journal of Robotics Research*, vol. 8, n. 4, pp. 92-112, 1989.

[7] R. Arkin, «Behavior-based Robot Navigation in Extended Domains,» *Journal of Adaptive Behavior*, vol. 1, n. 2, pp. 201-225, 1992.

[8] X. Chen e J. Wang, «Entropy-Based Crowd Evacuation Modeling With Seeking Behavior of Social Groups,» *IEEE Access*, vol. 9, pp. 4653-4664, 2021.

[9] N. S. Lachenmyer e S. Akasha, «Creating Life-like Autonomous Agents for Real-time Interactive,» in *ACM SIGGRAPH 2022 Talks*, New York, 2022.

[10] K. J. Yaxley, N. McIntyre, J. Park e J. Healey, «Sky Shepherds: A Tale of a UAV and Sheep,» in *Shepherding UxVs for Human-Swarm Teaming: An Artificial Intelligence Approach to Unmanned X Vehicles*, Abbass, Hussein A. and Hunjet, Robert A., 2021, pp. 189-206.

[11] R. F. Storms, C. Carere e C. K. Hemelrijk, «Complex patterns of collective escape in starling flocks under predation,» *Behavioral Ecology and Sociobiology*, vol. 73, n. 1, 2019.

[12] A. Spataru, L. C. Tranca, M. E. Penteliuc e M. Frincu, «Parallel Cloud Movement Forecasting based on a Modified Boids Flocking Algorithm,» in *20th International Symposium on Parallel and Distributed Computing*, 2021.

[13] S. Liu, L. Chen e J. Chai, «Public Opinion Evolution Based on the Flocking Algorithm,» in *2021 China Automation Congress (CAC)*, 2021.

[14] H. Y. Swathi, G. Shivakumar e H. S. Mohana, «Crowd Behavior Analysis: A Survey,» in *2017 International Conference on Recent Advances in Electronics and Communication Technology*, 2017.

[15] P. Remagnino, Li-Qun Xu, B. Zhan e D. N. Monekosso, «Crowd analysis: a survey,» *Machine Vision and Applications*, vol. 19, p. 345–357, 2008.

[16] M. A. B. M. Azahar, M. S. Sunar, D. Daman e A. Bade, «Survey on Real-Time Crowds Simulation,» in *Technologies for E-Learning and Digital Entertainment*, Berlino, Pan Zhigeng, Zhang Xiaopeng, El Rhalibi Abdennour, Woo Woontack, Li Yi, 2008, pp. 573-580.

[17] K. Ijaz, S. Sohail e S. Hashish, «A survey of latest approaches for crowd simulation and modeling using hybrid techniques,» in *Proceedings of the 2015 17th UKSIM-AMSS International Conference on Modelling and Simulation*, 2015.

[18] S. R. Musse, V. J. Cassol e D. Thalmann, «A history of crowd simulation: the past, evolution, and new perspectives,» *The Visual Computer*, vol. 37, n. 8, p. 3077–3092, 2021.

[19] D. Helbing e P. Molnár, «Social force model for pedestrian dynamics,» *PHYSICAL REVIEW E*, vol. 51, n. 5, pp. 4282-4286, 1995.

[20] S. R. Musse e D. Thalmann, «A Model of Human Crowd Behavior : Group Inter-Relationship and Collision Detection Analysis,» in *Computer Animation and Simulation '97*, Vienna, Springer, 1997, pp. 39-51.

[21] N. Farenc, S. R. Musse, E. Schweiss, M. Kallmann, O. Aune, R. Boulic e D. Thalmann, «A paradigm for controlling virtual humans in urban environment simulations,» *Applied Artificial Intelligence*, vol. 14, n. 1, pp. 69-91, 2000.

[22] D. Helbing, I. Farkas e T. Vicsek, «Simulating dynamical features,» *Nature*, vol. 407, n. 6803, pp. 487-490, 2000.

[23] C. Loscos e A. Meyer, «Intuitive crowd behavior in dense urban environments using local laws,» in *Theory and Practice of Computer Graphics, 2003. Proceedings*, Birmingham, 2003.

[24] M. Sung, L. Kovar e M. Gleicher, «Fast and accurate goal-directed motion synthesis for crowds,» in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, New York, 2005.

[25] R. Geraerts, A. Kamphuis, I. Karamouzas e M. Overmars, «Using the corridor map method for path planning for a large number of characters,» in *Motion in Games*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2008, pp. 11-22.

[26] K. Y. Wong e C. Loscos, «Hierarchical Path Planning for Virtual Crowds,» in *Motion in Games*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2008, pp. 43-50.

[27] A. Lerner, D. Lischinski e Y. Chrysanthou, «Crowds by Example,» *Computer Graphics Forum*, vol. 26, n. 3, pp. 655-664, 2007.

[28] B. G. Silverman, N. I. Badler, N. Pelechano e K. O'Brien, «Crowd simulation incorporating agent psychological models, roles and communication,» *V-Crowds*, vol. 43, pp. 21-30, date2005.

[29] N. Pelechano e N. Badler, «Modeling Crowd and Trained Leader Behavior during Building Evacuation,» *IEEE Computer Graphics and Applications*, vol. 26, n. 6, pp. 80-86, 2006.

[30] D. Thalmann e S. R. Musse, Crowd Simulation, Berlin, Heidelberg: Springer, 2007.

[31] N. Badler, J. Allbeck e N. Pelechano, Virtual Crowds: Methods, Simulation, and Control, Morgan and Claypool Publishers, 2008.

[32] F. Durupınar, N. Pelechano, J. M. Allbeck, U. Güdükbay e N. Badler, «How the Ocean Personality Model Affects the Perception of Crowds,» *IEEE Computer Graphics and Applications*, vol. 31, n. 3, pp. 22-31, 2009.

[33] T. B. Dutra, G. Priem, J. B. Cavalcante-Neto e C. Vidal, «Synthetic Vision-based Crowd Simulation: Reactive vs. Reactive Planning Approaches,» in *Proceedings of the 27th Conference on Computer Animation and Social Agents*, 2014.

[34] A. E. Başak, U. Güdükbay e F. Durupınar, «Using real life incidents for creating realistic virtual crowds with data-driven emotion contagion,» *Computers & Graphics*, vol. 72, pp. 70-81, 2018.

[35] E. Testa, R. C. Barros e S. R. Musse, «CrowdEst: a method for estimating (and not simulating) crowd evacuation parameters in generic environments,» *The Visual Computer*, vol. 35, p. 1119–1130 , 2019.

[36] V. J. Cassol, S. R. Musse, C. R. Jung e N. I. Badler, Simulating Crowds in Egress Scenarios, Springer, 2017.

[37] R.-C. Ho, S.-K. Wong, Y.-H. Chou, G.-W. Lin, T.-Y. Tsai, P.-H. Huang, Y.-S. Wang e I.-C. Yeh, «An authoring framework for time dependent crowd simulation,» in *SIGGRAPH ASIA 2016 Posters*,, New York, 2016.

[38] M. Moussaïd, M. Kapadia, T. Thrash, R. W. Sumner, M. Gross, D. Helbing e C. Hölscher, «Crowd behaviour during high-stress evacuations in an immersive virtual environment,» *Journal of the Royal Society Interface,* vol. 13, n. 122, 2016.

[39] F. Mirahadi e B. Y. McCabe, «EvacuSafe: A real-time model for building evacuation based on Dijkstra's algorithm,» *Journal of Building Engineering,* vol. 34, 2021.

[40] E. Hernandez-Orallo e A. Armero-Martínez, «How Human Mobility Models Can Help to Deal with COVID-19,» *Electronics,* vol. 10, n. 1, 2021.

[41] Y. Yuan, B. Goni-Ros, H. H. Bui, W. Daamen, H. L. Vu e S. P. Hoogendoorn, «Macroscopic pedestrian flow simulation using smoothed particle hydrodynamics (SPH),» *Transportation Research Part C: Emerging Technologies,* vol. 111, pp. 334-351, 2020.

[42] Z. Ren, P. Charalambous, J. Bruneau, Q. Peng e J. Pettré, «Group Modeling: A Unified Velocity-Based Approach,» *Computer Graphics Forum,* vol. 36, n. 8, pp. 45-56, 2017.

[43] W. T. Reeves, «Particle systems - a technique for modeling a class of fuzzy objects,» *Computer Graphics,* vol. 17, n. 3, p. 359–375, 1983.

[44] Processing, «Flocking, by Daniel Shiffman,» [Online]. Available: https://processing.org/examples/flocking.html. [Consultato il giorno 13 10 2022].

[45] R. Saeed, «Path Planning for Robot and Pedestrian Simulations,» UNICA IRIS Institutional Research Information System, Cagliari, 2021.

[46] I. Millington e J. Funge, Artificial Intelligence for Games, CRC Press, 2009.

[47] R. Dupre e V. Argyriou, «A human and group behavior simulation evaluation framework utilizing composition and video analysis,» *Computer Animation and Virtual Worlds,* vol. 30, n. 1, 2019.