



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA

Dipartimento di Informatica, Sistemistica e  
Comunicazione

Corso di Laurea Magistrale in Informatica

# **Analisi ed ottimizzazione della visibilità della segnaletica attraverso un ambiente di realtà virtuale e sistema multi-agente**

**Relatore:** Prof. Giuseppe Vizzari

**Tesi di Laurea Magistrale di:**

Emanuele Giannuzzi

Matricola 797819

**Anno Accademico 2023-2024**



# Abstract

Il presente lavoro si propone di sviluppare uno strumento per l'analisi e l'ottimizzazione della visibilità della segnaletica, sfruttando i dati forniti da modelli BIM (*Building Information Modeling*).

L'obiettivo principale è la creazione di un ambiente di realtà virtuale (VR) in cui simulare il flusso pedonale, permettendo di valutare il grado di visibilità e l'efficacia dei cartelli segnaletici.

Il sistema proposto integra diverse tipologie di analisi, tra cui valutazioni qualitative sulla visibilità della segnaletica, con l'obiettivo di ottimizzarne il posizionamento. Tali analisi considerano vari profili di utenti (ad esempio, persone di diversa altezza o in sedia a rotelle) e le diverse direzioni del loro movimento all'interno dello spazio simulato.

Oltre a fornire suggerimenti per migliorare il posizionamento della segnaletica, il sistema permette anche di valutare l'efficacia della segnaletica esistente, offrendo strumenti per testare differenti configurazioni di layout. In questo modo, si punta a rendere più efficienti i flussi pedonali, ridurre la confusione nelle aree affollate e migliorare l'esperienza degli utenti negli spazi pubblici o privati dove verrà implementato.



# Indice

<b>1</b>	<b>Introduzione</b>	<b>6</b>
<b>2</b>	<b>Informazioni preliminari</b>	<b>10</b>
2.1	Building Information Modeling . . . . .	10
2.2	Formato IFC . . . . .	11
<b>3</b>	<b>Stato dell'arte</b>	<b>15</b>
3.1	Valutazione della visibilità della segnaletica . . . . .	16
3.2	Simulazione del flusso pedonale . . . . .	17
3.2.1	Navigation Mesh . . . . .	17
3.2.2	Social Force Model . . . . .	21
<b>4</b>	<b>Costruzione dell'Ambiente</b>	<b>23</b>
4.1	Unity . . . . .	23
4.2	Import modello BIM . . . . .	24
4.3	Aree navigabili . . . . .	27
4.4	Piani di Visibilità e VCA . . . . .	30
4.4.1	Visibility Handler . . . . .	34
4.5	Generazione di segnaletica . . . . .	36
4.6	Marker e Routing . . . . .	37
4.6.1	Marker Generator . . . . .	39
<b>5</b>	<b>Simulazione del flusso pedonale</b>	<b>41</b>
5.1	Agenti e ambiente . . . . .	41
5.2	Architettura degli agenti . . . . .	42
5.2.1	NavMesh Agent . . . . .	43
5.2.2	Social Force Agent . . . . .	43
5.2.3	Agent . . . . .	46
5.2.4	Markers Aware Agent . . . . .	47
5.2.5	Signboard Aware Agent . . . . .	47
5.2.6	Simulation Agent . . . . .	48

5.2.7	Wanderer Agent . . . . .	49
5.2.8	Routed Agent . . . . .	53
5.3	Introduzione degli agenti nell’ambiente . . . . .	53
<b>6</b>	<b>Validazioni</b>	<b>57</b>
6.1	Validazione del Social Force Agent . . . . .	57
6.1.1	Confronto tra i risultati . . . . .	58
6.2	Validazione Wanderer Agent . . . . .	61
6.2.1	Agenti di confronto . . . . .	61
6.2.2	Confronto delle performance degli agenti . . . . .	62
<b>7</b>	<b>Valutazione della segnaletica</b>	<b>65</b>
7.1	Analisi Statica . . . . .	66
7.2	Ricerca dell’ottimo . . . . .	70
7.2.1	Rete di marker e percorso ottimale . . . . .	70
7.2.2	Parametri della simulazione . . . . .	71
7.2.3	Risultati . . . . .	72
<b>8</b>	<b>Conclusioni</b>	<b>76</b>
8.1	Sviluppi futuri . . . . .	77

# Capitolo 1

## Introduzione

Nella vita quotidiana, siamo costantemente circondati da diversi tipi di segnaletica, dai segnali stradali alle indicazioni presenti all'interno di edifici complessi come uffici o centri commerciali. Questi segnali rivestono un ruolo essenziale nell'aiutare le persone a orientarsi, segnalare pericoli e fornire istruzioni chiare per spostamenti sicuri ed efficienti.

Nonostante la loro importanza, i cartelli spesso non risultano efficaci come dovrebbero. Posizionamenti inappropriati, dimensioni ridotte, colori poco visibili e ostacoli fisici possono ridurre la leggibilità dei segnali, causando disorientamento e aumentando il rischio di confusione o incidenti. Questo progetto si propone di affrontare tali problematiche sviluppando uno strumento open-source per analizzare e ottimizzare la visibilità della segnaletica all'interno degli edifici, sfruttando le informazioni contenute in un modello BIM (Building Information Modeling). Migliorare la visibilità e la leggibilità dei cartelli non solo facilita la navigazione, ma aumenta anche la sicurezza in ambienti complessi o affollati.

Il codice sorgente del progetto è disponibile nella repository GitHub Signposting-Multi-Agent-Simulation<sup>1</sup> ed è rilasciato sotto licenza GPL-3.0. Questa licenza garantisce che il software sia libero e aperto, permettendo a chiunque di modificarlo e ridistribuirlo, a patto che qualsiasi modifica o software derivato mantenga la stessa licenza, assicurando così che le versioni future del progetto rimangano open-source.

---

<sup>1</sup><https://github.com/EmanueleGiannuzzi/Signposting-Multi-Agent-Simulation>

## Obiettivi del progetto

L'obiettivo principale di questo progetto è sviluppare uno strumento capace di analizzare e ottimizzare la visibilità della segnaletica all'interno degli edifici, tenendo conto delle variabili sopra citate. La soluzione proposta sfrutterà le informazioni contenute nel modello BIM dell'edificio per simulare il flusso di pedoni e determinare l'efficacia dei cartelli presenti, identificando eventuali criticità.

Invece di puntare a una soluzione unica per ogni singolo edificio, lo strumento sviluppato è concepito per offrire una serie di moduli altamente configurabili. Questo approccio permette ai professionisti del settore di adattare i parametri del sistema alle specifiche esigenze di ciascun ambiente, garantendo al contempo un'interfaccia intuitiva per una rapida implementazione.

È importante sottolineare che il progetto **non si occupa di scenari di emergenza** né di cartelli di emergenza, che richiederebbero un'analisi e una modellazione molto più complessa, poiché coinvolgono dinamiche di evacuazione che esulano dall'ambito di questo studio. Lo scopo è concentrarsi su una gestione ordinaria della segnaletica informativa per facilitare la navigazione quotidiana.

## Sfide Tecniche

Per raggiungere questi obiettivi, è necessario affrontare diverse sfide complesse. Innanzitutto, è necessario comprendere quali fattori influenzano l'efficacia della segnaletica, come il posizionamento corretto, le dimensioni ottimali e il contenuto visibile in varie condizioni. Ogni edificio ha caratteristiche uniche e non esiste una soluzione universale che funzioni in tutti i contesti.

In aggiunta, la geometria dell'edificio può influenzare profondamente la visibilità dei cartelli. Elementi architettonici come muri, scale e arredi possono ostruire la vista dei segnali, rendendo difficile un'analisi accurata senza considerare questi ostacoli. Nei contesti ad alta densità pedonale, diventa ancora più complesso, poiché il flusso di persone in movimento influenza la capacità di notare e seguire i cartelli.

Un'altra sfida cruciale è la diversità degli utenti. Le persone che attraversano un edificio hanno diverse altezze e campi visivi: adulti, bambini e persone su sedia a rotelle non percepiscono i segnali allo stesso modo. Questo aggiunge ulteriore complessità nella definizione di criteri di visibilità che siano adatti a tutte le categorie di utenti.

Infine, durante lo sviluppo, particolare attenzione è stata posta all'architettura del progetto, che è stata progettata seguendo i principi di **Clean Architecture** [13], come descritti da Robert C. Martin. Questo approccio garantisce che il codice sia modulare, facilmente espandibile e mantenibile, semplificando l'integrazione di nuove funzionalità o studi futuri. Tale architettura rende inoltre lo strumento flessibile e accessibile ad altri sviluppatori che desiderano estenderlo o adattarlo a nuovi scenari.

## Struttura della tesi

### Capitolo 1 - Introduzione

Viene introdotto il tema centrale della tesi, ovvero l'importanza della segnaletica negli edifici e le sfide che ne derivano. Si presenta inoltre l'obiettivo del progetto e la metodologia utilizzata, con un focus sugli strumenti di simulazione sviluppati.

### Capitolo 2 - Informazioni Preliminari

Questo capitolo fornisce una panoramica dei concetti di base e delle tecnologie utilizzate nel progetto, come il modello BIM, la navigazione multi-agente e la rappresentazione degli edifici tramite IFC (Industry Foundation Classes). Questi elementi formano le fondamenta tecniche della tesi.

### Capitolo 3 - Stato dell'Arte

Vengono presentati e discussi i lavori e gli studi esistenti nel campo della visibilità della segnaletica e della simulazione del flusso pedonale. Si approfondiscono le tecniche attuali, evidenziando i punti di forza e le limitazioni delle soluzioni precedenti.

### Capitolo 4 - Costruzione dell'Ambiente

In questo capitolo, viene descritta la creazione dell'ambiente di realtà virtuale, dall'importazione del modello alla definizione delle aree navigabili e alla generazione della segnaletica. Si discutono i moduli sviluppati per la gestione degli agenti e della segnaletica all'interno del modello.

### Capitolo 5 - Simulazione del Flusso Pedonale

Si entra nel dettaglio della simulazione del comportamento degli agenti, descrivendo l'architettura sviluppata e i vari tipi di agenti utilizzati. Vengono inoltre presentate

le strategie per l'ottimizzazione del movimento e per l'interazione degli agenti con la segnaletica.

## **Capitolo 6 - Validazioni**

Questo capitolo presenta le validazioni eseguite per confrontare i risultati delle simulazioni con i dati provenienti da modelli empirici o studi precedenti. Vengono discussi i risultati ottenuti e il confronto con i modelli originali da cui alcuni componenti sono stati derivati.

## **Capitolo 7 - Valutazione della Segnaletica**

Qui vengono eseguiti test pratici sulla segnaletica presente nell'edificio simulato. Si esaminano sia le performance degli agenti che l'efficacia dei posizionamenti dei cartelli, per fornire una valutazione qualitativa e quantitativa del loro impatto.

## **Capitolo 8 - Conclusioni**

Il capitolo conclusivo riassume i risultati raggiunti e offre una riflessione sulle potenzialità future dello strumento sviluppato.

# Capitolo 2

## Informazioni preliminari

Prima di procedere con la descrizione del progetto sviluppato, è essenziale introdurre alcuni concetti fondamentali che saranno alla base delle sezioni successive. In particolare, familiarizzeremo con il concetto di **Building Information Modeling** (BIM) e il formato **IFC** (Industry Foundation Classes), che costituiscono il cuore del modello di gestione dell'edificio utilizzato nel progetto.

Questo capitolo servirà a chiarire come questi strumenti tecnologici siano stati utilizzati per integrare la segnaletica all'interno dei modelli digitali di edifici e come siano stati adattati per permettere l'analisi della visibilità e della navigazione pedonale negli ambienti simulati.

### 2.1 Building Information Modeling

Il *Building Information Modeling* (BIM) è un processo [2] che fa uso di una vasta gamma di strumenti, per generare e gestire rappresentazioni dettagliate di edifici e infrastrutture. Il suo obiettivo è quello di coprire l'intero ciclo di vita di un progetto edilizio, che va ben oltre la semplice rappresentazione tridimensionale della struttura.

Ogni componente dell'edificio, dalle pareti agli impianti, è rappresentato in maniera virtuale e contiene informazioni sulle sue caratteristiche fisiche, funzionali e temporali.[6]

Questo modello digitale è in grado di fornire una visione complessiva e interconnessa del progetto, favorendo una maggiore collaborazione tra i professionisti coinvolti. In questo contesto, è importante considerare come le informazioni relative alla segnaletica si possano integrare nel processo BIM, contribuendo a una gestione più efficiente degli spazi.

Nel linguaggio tecnico e nell'uso comune, il termine **BIM** viene impiegato come meto-

nimia per riferirsi non solo al processo di *Building Information Modeling*, ma anche al modello digitale risultante.

In questo documento, per semplificare il discorso, verrà fatto uso del termine **BIM** per indicare il modello informativo tridimensionale.

## 2.2 Formato IFC

I software per la gestione di **BIM** disponibili sul mercato utilizzano una grande varietà di formati. Tra i più diffusi si possono citare RVT (utilizzato da Autodesk Revit), DWG (impiegato in AutoCAD), e SKP (il formato di SketchUp).

Tuttavia, la maggior parte di questi formati sono proprietari, limitando l'interoperabilità e creando difficoltà nell'utilizzo dei dati su piattaforme diverse da quelle native.

Per questo motivo è stato scelto di basare il progetto su file IFC (*Industry Foundation Classes*), ovvero un formato aperto, non proprietario e riconosciuto come standard internazionale (*ISO 16739*). [11]

Il formato IFC promuove funzionalità agnostiche rispetto all'hardware e al software, garantendo la compatibilità con la maggior parte degli strumenti più utilizzati nel settore.[18]

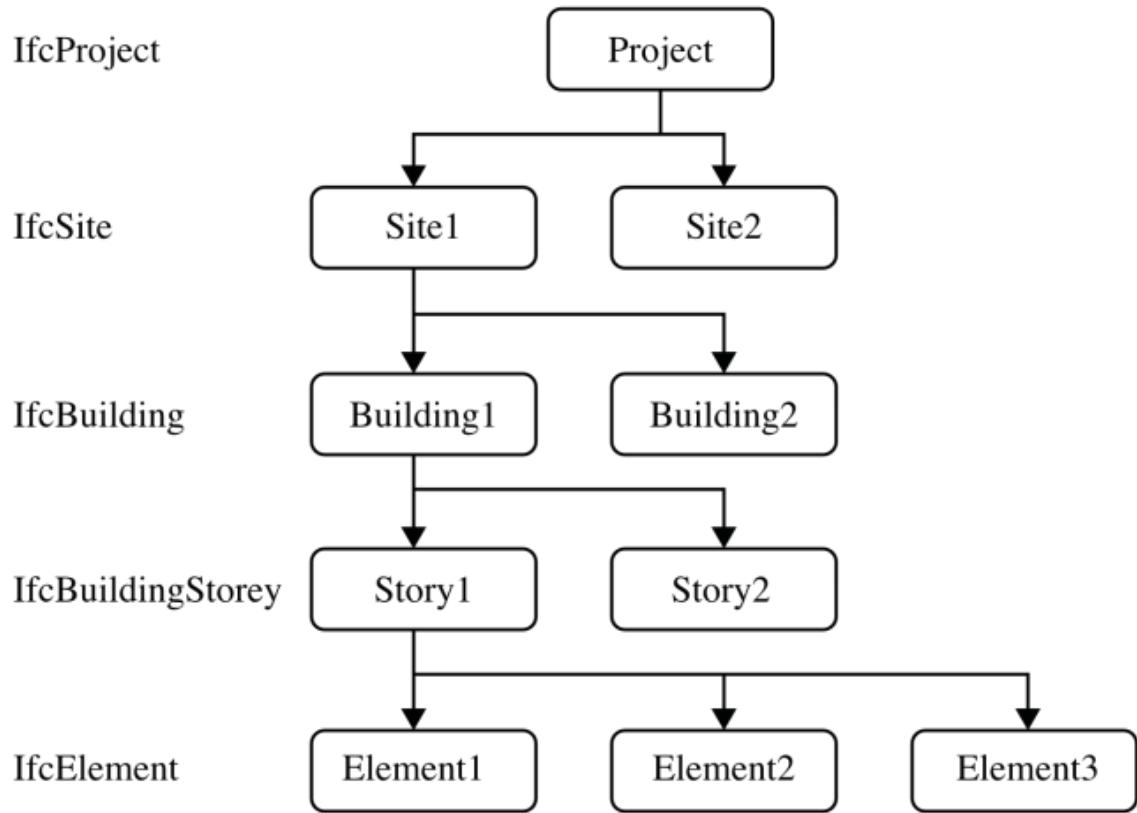


Figura 2.1: Esempio gerarchia degli elementi in un file IFC. Fonte: [17]

Ciascuna entità all'interno del file IFC è associata a un tag specifico, indicativo della parte dell'edificio che rappresenta. Questi tag includono, ad esempio, IfcBuilding, IfcStorey, IfcDoor, riflettendo la natura delle diverse componenti della struttura.

Oltre all'assegnazione di tag, è possibile stabilire relazioni tra diverse entità e attribuire loro proprietà, come ad esempio il materiale o il colore, contribuendo a una rappresentazione più completa e dettagliata del modello dell'edificio. (Figura 2.2)

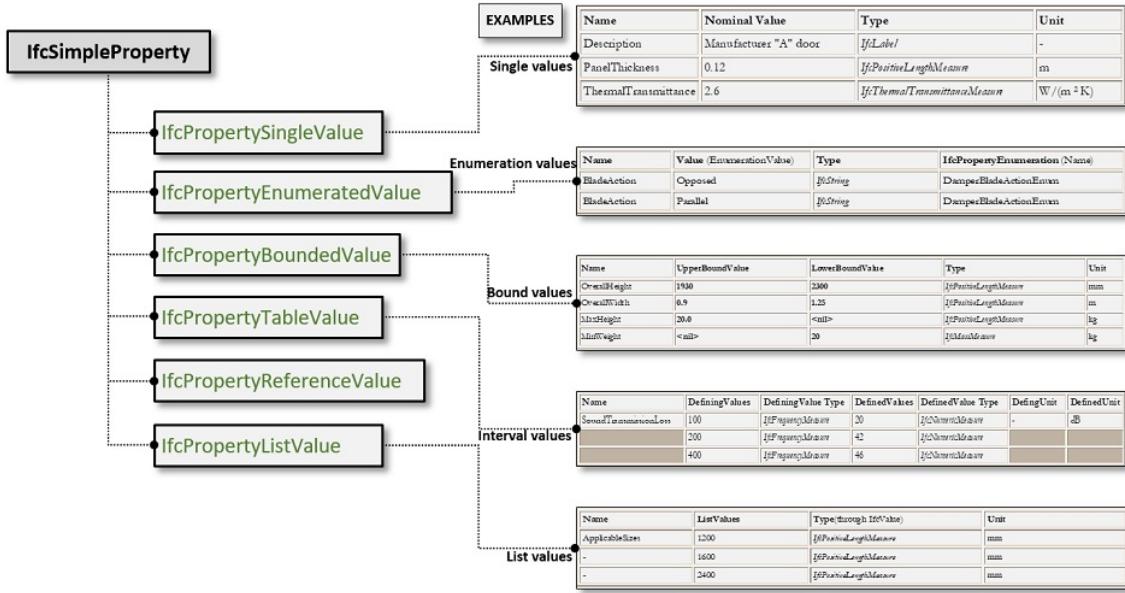


Figura 2.2: Esempio di proprietà assegnabili ad un entità nello schema IFC. Fonte: [1]

Un aspetto importante che rende il formato IFC ideale per il nostro progetto è la possibilità di definire nuovi tag.

Questa caratteristica permette di adattare il modello BIM alle esigenze specifiche, come nel nostro caso la definizione del tag **IfcSign**. Questo tag è stato creato per rappresentare cartelli o insegne contenenti informazioni utili per gli occupanti dell'edificio.

Ogni entità contrassegnata con il tag **IfcSign** deve avere un insieme di proprietà associate.

Queste proprietà sono definite dal tag **Pset\_SignboardCommon**, contenente i seguenti elementi:

- ViewingDistance
- ViewingAngle
- MinimumReadingTime

Queste caratteristiche sono intrinseche al cartello stesso e vengono definite a priori. Esse forniscono informazioni fondamentali per comprendere come il cartello sarà percepito dagli utenti, e saranno oggetto di un'analisi dettagliata nei capitoli successivi.

Nonostante il riconoscimento a livello internazionale come standard, il formato IFC non è ancora di uso comune a livello globale.

I formati proprietari, essendo più difficili da distribuire, aggiungono un livello di protezione desiderabile quando il costo dei beni digitali diventa elevato.

Va sottolineato che il formato IFC è molto utilizzato per lo scambio di modelli tra diversi software BIM; di conseguenza, la sua flessibilità e la possibilità di essere importato ed esportato dalla maggior parte dei software, rendono il formato IFC ideale per il nostro scopo.

# Capitolo 3

## Stato dell'arte

Negli ultimi anni, la ricerca sull'interazione tra pedoni e segnaletica negli edifici ha acquisito un'importanza crescente, soprattutto in contesti come aeroporti, ospedali, centri commerciali e altri spazi pubblici ad alta densità. La segnaletica svolge un ruolo cruciale nel guidare le persone attraverso questi ambienti complessi, garantendo che trovino rapidamente la loro destinazione e che i flussi di persone siano organizzati in modo efficiente. Tuttavia, affinché un cartello sia realmente efficace, esso deve essere sia visibile sia comprensibile, indipendentemente dalle caratteristiche fisiche dell'ambiente e dei suoi occupanti.

La visibilità della segnaletica è influenzata da molteplici fattori, inclusi quelli fisici (posizione, orientamento, dimensioni) e cognitivi (leggibilità, tempo di reazione dell'osservatore). Pertanto, è necessario adottare modelli e metodologie che permettano di valutare in modo accurato la capacità dei cartelli di trasmettere le informazioni desiderate ai pedoni in movimento all'interno degli edifici.

Oltre agli studi teorici sulla visibilità statica dei cartelli, la simulazione dinamica del flusso pedonale ha mostrato di essere uno strumento essenziale per comprendere come le persone interagiscano con la segnaletica in ambienti reali.

Attraverso la simulazione, è possibile esaminare fattori come la densità dei pedoni, la velocità di movimento ed i percorsi percorsi seguiti, in modo da capire come questi vengono influenzati dalle indicazioni.

In questo capitolo, verranno esaminati i principali approcci alla valutazione della visibilità della segnaletica e i modelli di simulazione dei pedoni utilizzati per analizzare il comportamento degli utenti in ambienti complessi. In particolare, ci concentreremo su due aspetti fondamentali: la determinazione delle caratteristiche fisiche che influenzano la visibilità della segnaletica e l'uso di simulazioni dinamiche per verificare l'efficacia della segnaletica in situazioni reali.

### 3.1 Valutazione della visibilità della segnaletica

La valutazione della visibilità di un cartello è un passaggio cruciale per garantire che le informazioni in esso contenuto siano efficacemente recepite dagli utenti dell'edificio. In questo contesto, diversi parametri devono essere considerati per determinare la percezione corretta della segnaletica, tra cui fattori di spazio, ambientali e umani.

Un contributo significativo in questo ambito è stato fornito da Filippidis et al. [7], i quali hanno proposto un approccio innovativo, basato sulla definizione di una regione di spazio chiamata *Visibility Catchment Area* (VCA), di cui possiamo vedere un esempio in Figura 3.1. La VCA rappresenta l'area entro cui un osservatore, rivolto verso il cartello, è in grado di acquisire visivamente le informazioni contenute al suo interno.

Questo concetto è stato successivamente adottato ed ampliato da una grande quantità di studi, tra cui la sezione “*Signage visibility analysis and optimization system using BIM-enabled virtual reality (VR) environments*” in “Advanced Engineering Informatics”[15], su cui questo lavoro si basa.

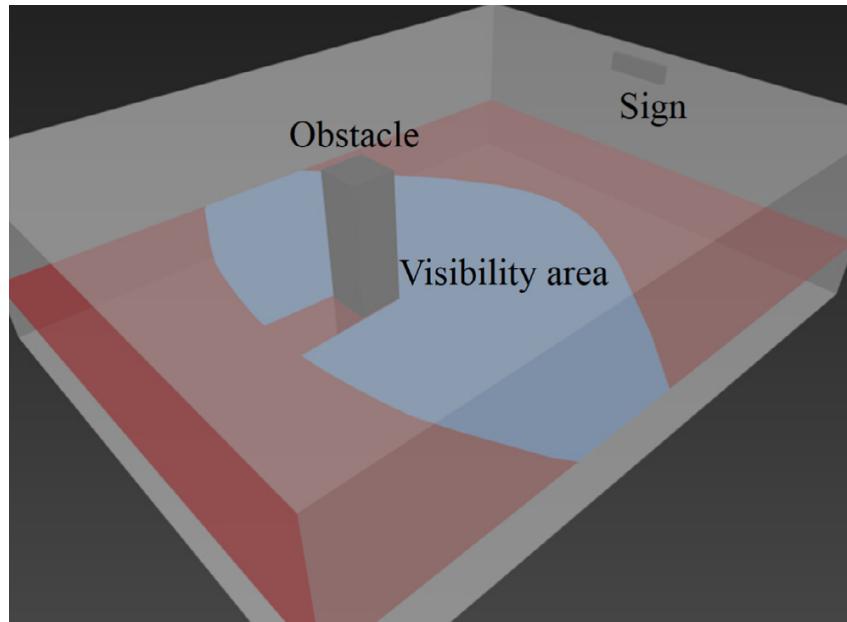


Figura 3.1: Esempio di VCA in blu. Fonte: [15]

Il processo di costruzione della VCA considera vari parametri, come la posizione e l'altezza del cartello rispetto al pavimento, il suo orientamento rispetto all'osservatore e la presen-

za di eventuali ostacoli che potrebbero ostruirne la visibilità. Tuttavia questo approccio presenta alcuni limiti, in quanto si presume che l'individuo sia attento e in grado di elaborare le informazioni all'interno del proprio campo visivo, indipendentemente dalla loro posizione.

In sintesi, un cartello è considerato visibile e leggibile da un osservatore se quest'ultimo si trova all'interno della sua *area di visibilità* (VCA) e il cartello si trova nel campo visivo dell'osservatore.

## 3.2 Simulazione del flusso pedonale

La visibilità statica di un cartello, come definita dalla VCA, non è sufficiente a garantire che esso sia effettivamente percepito e compreso dagli utenti dell'edificio. Esso potrebbe trovarsi in una zona poco trafficata o essere invisibile a certe categorie di soggetti, come, ad esempio, una persona in sedia a rotelle.

Per questo motivo, è necessario simulare il flusso di pedoni per verificare come diverse categorie di individui interagiscono con la segnaletica in ambienti dinamici.

L'obiettivo principale della simulazione è raccogliere dati sulla reale visibilità della segnaletica da parte di diversi tipi di persone.

Un sistema di simulazione efficace deve soddisfare i seguenti requisiti:

- Leggerezza: Il modello deve supportare simulazioni in tempo reale o accelerate senza degradazioni nelle prestazioni.
- Robustezza: Deve gestire correttamente un numero elevato di agenti senza compromettere la precisione.
- Flessibilità: Il modello deve adattarsi a un ambiente tridimensionale complesso, tenendo conto di ostacoli, muri e dislivelli.

Nei paragrafi successivi verranno valutate diverse opzioni considerate.

### 3.2.1 Navigation Mesh

Il concetto di *Navigation Mesh* (NavMesh) è uno strumento ampiamente utilizzato nel campo della simulazione e della navigazione automatizzata in ambienti virtuali. Si tratta di una rappresentazione geometrica dello spazio di navigazione, suddiviso in una rete di

poligoni che definiscono le aree percorribili da un agente.

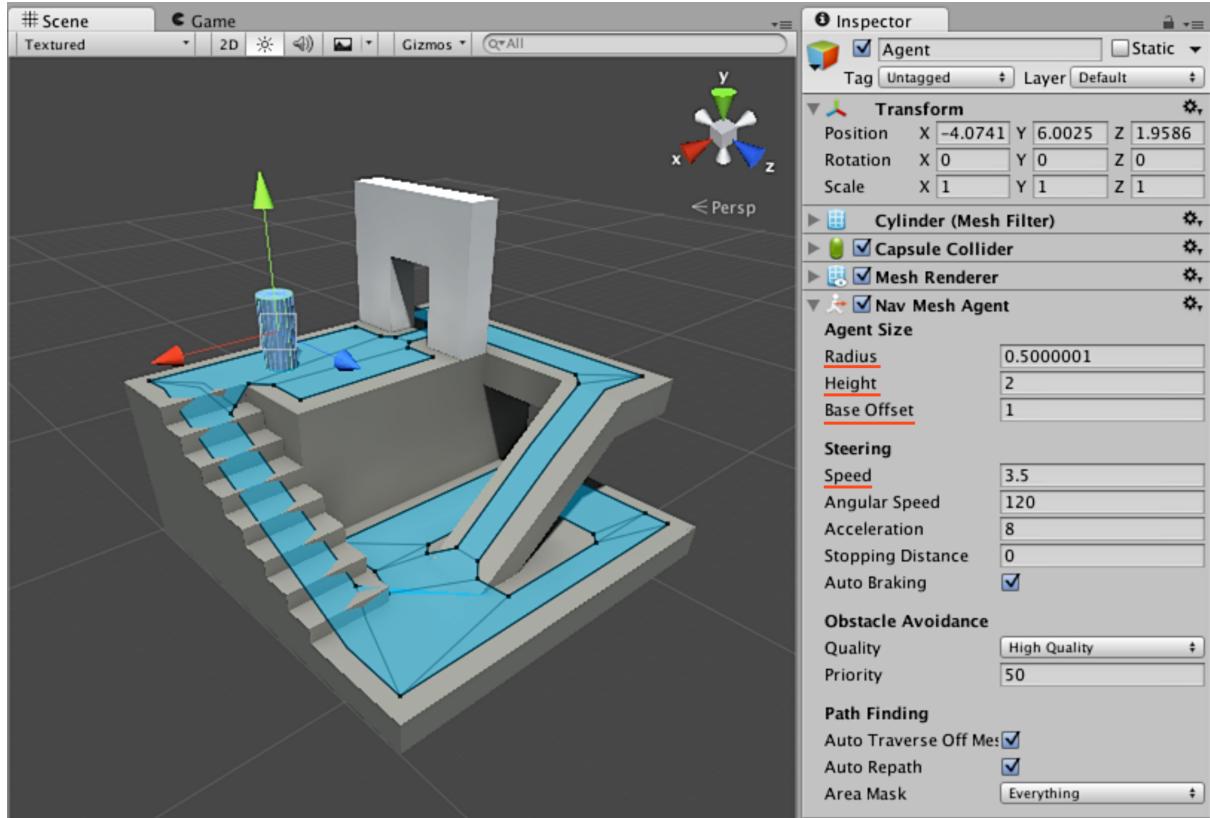


Figura 3.2: Esempio di *Navigation Mesh* che si estende su due livelli.  
Fonte: [20]

Le NavMesh permettono agli agenti di navigare in modo efficiente all'interno di uno spazio, evitando ostacoli statici e muovendosi da un punto all'altro tramite percorsi ottimizzati.

Questo sistema è particolarmente utile in scenari tridimensionali complessi, come edifici multi-piano o ambienti urbani, dove la navigazione deve tenere conto di elementi come scale, dislivelli, corridoi stretti e ostacoli architettonici. Essi consentono agli agenti di muoversi attraverso lo spazio simulato seguendo percorsi calcolati dinamicamente, in modo da ottenere i tragitti migliori per raggiungere una destinazione specifica.

Uno dei principali vantaggi dei NavMesh è la loro efficienza: una volta che la rete di navigazione è stata generata, il calcolo dei percorsi è molto rapido, il che li rende ideali per applicazioni in tempo reale. Questo è particolarmente importante in simulazioni con un elevato numero di agenti, come quelle che coinvolgono flussi pedonali all'interno di edifici affollati, dove ogni agente deve interagire con gli altri e con l'ambiente circostante in modo dinamico.

L'implementazione degli agenti NavMesh, tuttavia, può presentare alcune limitazioni. In scenari dove gli ostacoli o le condizioni cambiano frequentemente, come nel caso di flussi pedonali densi o contesti con elevata interazione tra agenti, essi possono rivelarsi insufficienti. Infatti, non sono progettati per gestire in modo ottimale situazioni complesse dove il comportamento collettivo dei pedoni e i flussi direzionali possono generare congestioni o collisioni.

Questo è un limite significativo quando si cerca di simulare il comportamento umano in modo più realistico, come nel caso della visibilità della segnaletica in ambienti dinamici.

Per testare l'efficacia di NavMesh nella simulazione di pedoni, è stato utilizzato l'omonimo sistema di navigazione multi-agente offerto dalla piattaforma Unity<sup>1</sup>.

Questo strumento, è ampiamente usato, specialmente in ambito videoludico, per la navigazione di agenti in ambienti virtuali, grazie alla sua flessibilità e leggerezza.

Unity offre ne un'implementazione generica che permette agli agenti di muoversi all'interno di ambienti tridimensionali complessi, e include un sistema di *obstacle avoidance* per evitare collisioni con altri agenti e ostacoli statici.

Nella fase di test, il modello degli agenti NavMesh di Unity ha mostrato ottime prestazioni in simulazioni con flussi pedonali moderati.

Il sistema è in grado di supportare un numero elevato di agenti simultaneamente, senza compromettere la fluidità della simulazione. Tuttavia, quando sono stati introdotti scenari con flussi pedonali molto densi e movimenti controcorrente, sono emersi dei limiti. Gli agenti mostravano sovrapposizioni e movimenti erratici, soprattutto in prossimità di ostacoli o in situazioni di congestione. Questi comportamenti non realistici compromettono la capacità del sistema di simulare accuratamente l'interazione dei pedoni con la segnaletica.

I seguenti test mostrano esempi delle limitazioni del sistema NavMesh di Unity in condizioni di traffico pedonale sempre più impegnativo. In questa relazione verranno riportate solo delle immagini rappresentative, ma sarà possibile cliccare sul link per vedere i test nella loro interezza.

---

<sup>1</sup>Piattaforma nata per lo sviluppo di videogiochi. Maggiori informazioni nel capitolo 4.1

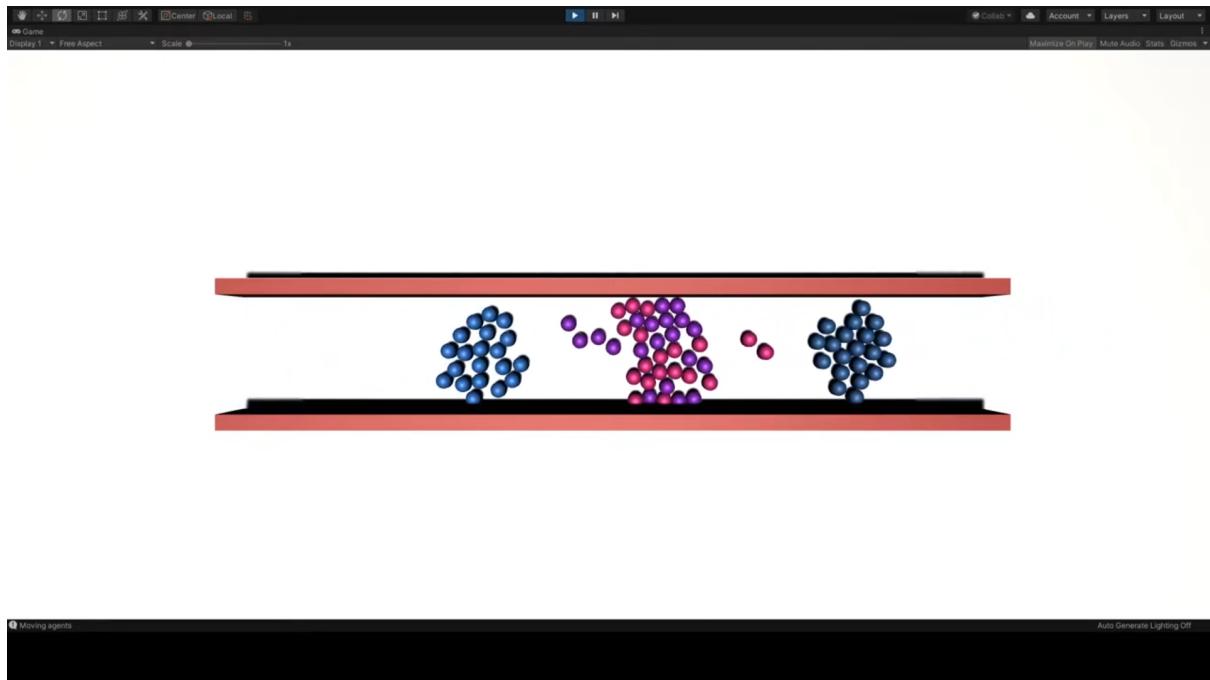


Figura 3.3: Test 1: Gruppi di agenti attraversano un corridoio in direzione opposta. <https://www.youtube.com/watch?v=08RZ2dKRbTs>

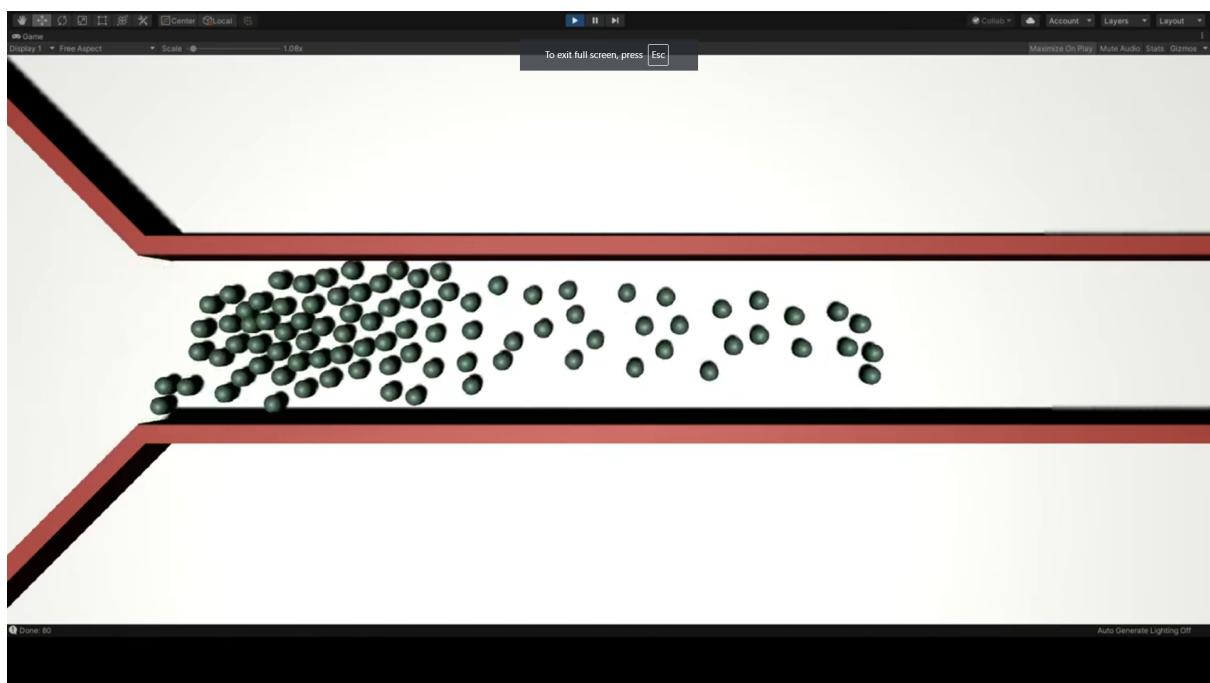


Figura 3.4: Test 2: Da 10 a 100 agenti che passano attraverso una strettoia. <https://www.youtube.com/watch?v=PcSm7nMf1Ao>

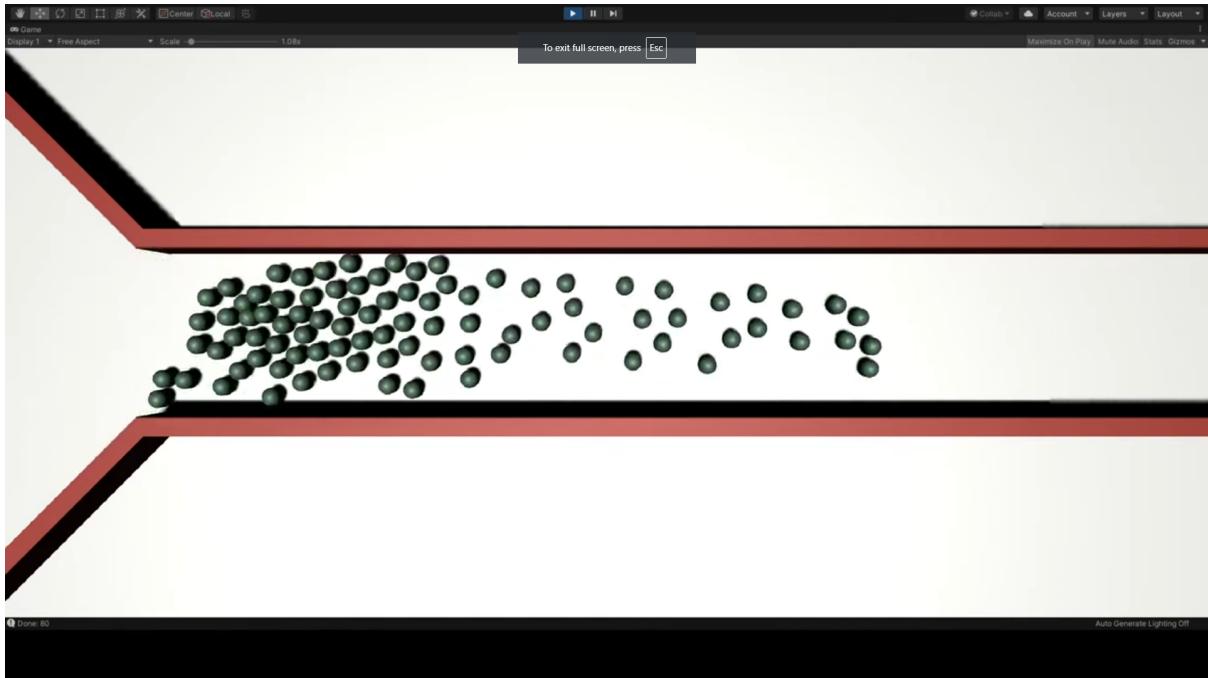


Figura 3.5: Test 3: Due gruppi di 1000 agenti attraversano un corridoio in direzione opposta. Questo test rappresenta condizioni estreme, ma rende molto evidenti i limiti dello strumento fornito da Unity.  
<https://www.youtube.com/watch?v=0ackMvqqVa8>

Nonostante gli agenti NavMesh di Unity rappresentino una soluzione efficiente e flessibile per la simulazione della navigazione pedonale in molti contesti, le simulazioni in situazioni di traffico pedonale elevato richiedono modelli più avanzati che possano gestire meglio l’interazione tra gli agenti e le condizioni di flusso dinamico.

### 3.2.2 Social Force Model

Volendosi distaccare da un modello che possiamo considerare “*naive*”, come NavMesh, sono state considerate diverse opzioni. Molte di queste soluzioni si sono rivelate inadeguate poiché non applicabili a spazi continui e tridimensionali, oppure troppo complesse e dispendiose in termini di performance.

Alcuni modelli, ad esempio, erano ottimizzati per scenari di evacuazione in situazioni di emergenza, mentre in questo lavoro preferiamo concentrarci esclusivamente su flussi pedonali ordinari e su come questi influenzano la percezione dei cartelli.

Uno dei modelli più efficaci per simulare il comportamento pedonale in questi contesti è il *Social Force Model* [9], introdotto da Helbing e Molnár nel 1995.

Questo modello si basa sull’idea che i pedoni siano influenzati da “forze sociali”<sup>2</sup> che de-

---

<sup>2</sup>Stimolo o impulso che porta ad un’azione sociale

terminano il loro movimento: una forza di attrazione verso la destinazione, una forza di repulsione per evitare collisioni con ostacoli e altri pedoni, e, in alcuni casi, una forza di coesione che li mantiene vicini ai membri del gruppo.

Queste interazioni generano comportamenti dinamici, come la formazione spontanea di corsie o l'auto-organizzazione in flussi opposti di pedoni.

Uno studio fondamentale in questo ambito è "*Experimental study of the behavioural mechanisms underlying self-organization in human crowds*" [16], che dimostra come le forze sociali portino alla formazione spontanea di pattern organizzati nelle folle.

Questi fenomeni emergenti sono centrali per comprendere il comportamento pedonale in ambienti ad alta densità, rendendo il modello particolarmente adatto alla simulazione di flussi pedonali complessi, dove l'interazione tra gli individui influisce significativamente sulle traiettorie.

Nel contesto della segnaletica, il *Social Force Model* risulta particolarmente utile per simulare come i pedoni interagiscono con i cartelli in ambienti affollati.

La visibilità di un cartello non dipende esclusivamente dalla sua posizione fisica, ma anche da fattori come la densità del flusso pedonale e le traiettorie dinamiche dei pedoni.

In situazioni di traffico intenso, la visibilità di un cartello può essere temporaneamente ostruita dal passaggio di altri pedoni, o il percorso di un osservatore può essere deviato dalla necessità di evitare collisioni, alterando così la sua possibilità di notare un cartello.

Il principale vantaggio del modello delle forze sociali è la capacità di simulare comportamenti collettivi complessi e realistici, che emergono naturalmente in contesti affollati. Rispetto ai modelli più semplici, come NavMesh, il *Social Force Model* tiene conto delle interazioni dinamiche tra gli individui, rendendolo più accurato per simulare flussi pedonali in ambienti reali. Bisogna però notare che questo approccio può risultare computazionalmente oneroso, soprattutto in simulazioni su larga scala, e richiede una calibrazione accurata dei parametri per garantire un comportamento accettabile.

# Capitolo 4

## Costruzione dell'Ambiente

### 4.1 Unity

Per la costruzione dell'ambiente virtuale, è fondamentale utilizzare uno strumento particolarmente flessibile, in grado di gestire modelli tridimensionali e fornire un linguaggio di programmazione ad alto livello che offre un'infrastruttura solida per la programmazione delle dinamiche di simulazione. Tali requisiti vengono pienamente soddisfatti dai moderni *game engine*, piattaforme originariamente impiegate solo nello sviluppo di videogiochi, ma che negli ultimi anni si sono dimostrate estremamente versatili per applicazioni in vari settori, come l'architettura, l'ingegneria e persino l'industria cinematografica.

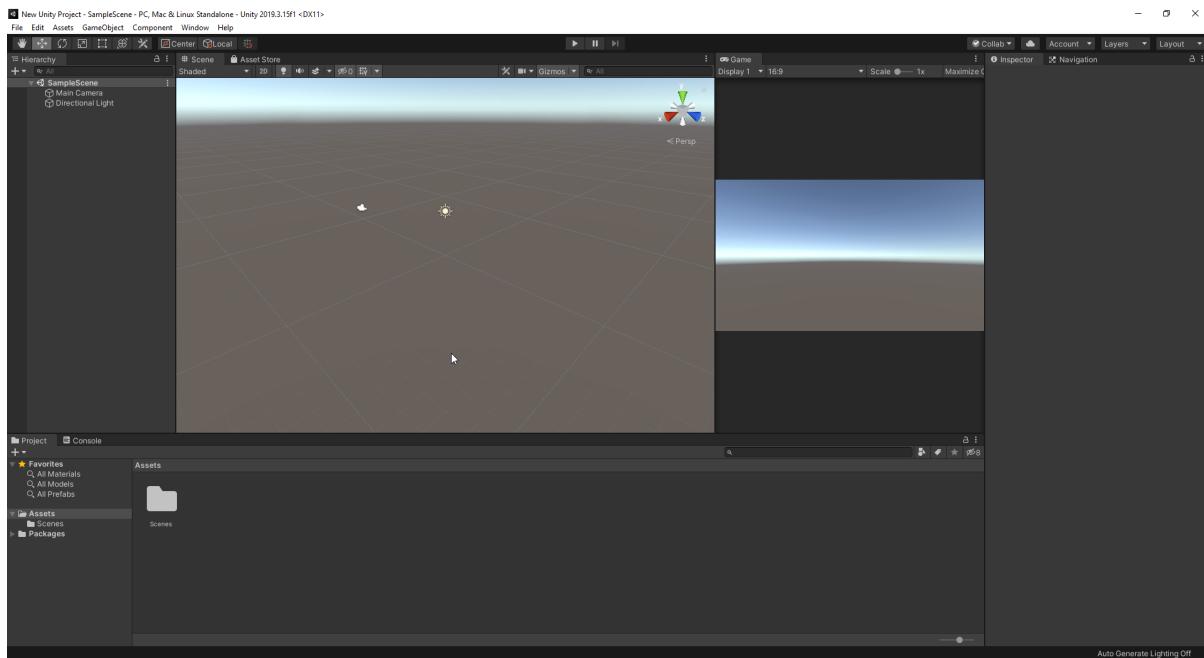


Figura 4.1: Come si presenta l'Editor di Unity quando viene creato un nuovo progetto

Il *game engine* scelto per questo progetto è *Unity*, nella sua versione 2022.3.26f1 LTS. *Unity* è stato selezionato principalmente per la sua capacità di gestire ambienti 3D in modo fluido e per la vasta gamma di strumenti *out-of-the-box* che mette a disposizione. Un altro vantaggio rilevante di *Unity* è l'adozione di C# come linguaggio di programmazione principale. C#, un linguaggio di programmazione sviluppato da Microsoft, è ampiamente apprezzato per la sua robustezza e per il supporto a un'ampia gamma di feature, tra la gestione di *eventi*, di cui verrà fatto largo uso in modo da ottenere un architettura del codice pulita.

Durante la fase di scelta della piattaforma, sono stati valutati anche altri motori grafici come *Unreal Engine* e *Godot*.

Il primo è noto per la sua potenza grafica e le capacità di rendering fotorealistico, mentre *Godot* sta crescendo in popolarità grazie alla sua natura open-source e alla flessibilità che offre a sviluppatori indipendenti.

Tuttavia, al momento dell'inizio dello sviluppo, *Unity* rappresentava la scelta più logica. La disponibilità di una documentazione dettagliata e una vasta comunità di supporto, ha semplificato notevolmente la fase di apprendimento e sperimentazione. Essa è infatti una delle più complete nel settore, con tutorial, esempi pratici e forum di discussione attivi, il che ha facilitato la risoluzione di problemi tecnici durante lo sviluppo.

Entrambi i concorrenti rappresentano ora opzioni valide per il futuro, soprattutto man mano che le loro comunità continuano a crescere e migliorare.

## 4.2 Import modello BIM

*Unity* non supporta nativamente il formato IFC (Industry Foundation Classes), discusso nel Capitolo 2.2. Per gestire i modelli BIM in *Unity*, è stato sviluppato un modulo dedicato, denominato **IfcOpenShellParser**, che si occupa dell'intero processo di conversione del modello e la gestione dei relativi dati all'interno dell'ambiente di simulazione.

Esso fa uso della libreria open-source *IfcOpenShell*<sup>1</sup>, che permette di trasformare i file IFC in tre formati distinti compatibili con *Unity* e poi associare i dati dello schema IFC agli elementi del modello 3D.

*IfcOpenShell* esegue la conversione del file IFC nei seguenti formati:

- **OBJ**: contiene le informazioni sulla geometria dell'edificio
- **MTL**: include i dati relativi ai materiali, ad esempio colore ed opacità delle superfici

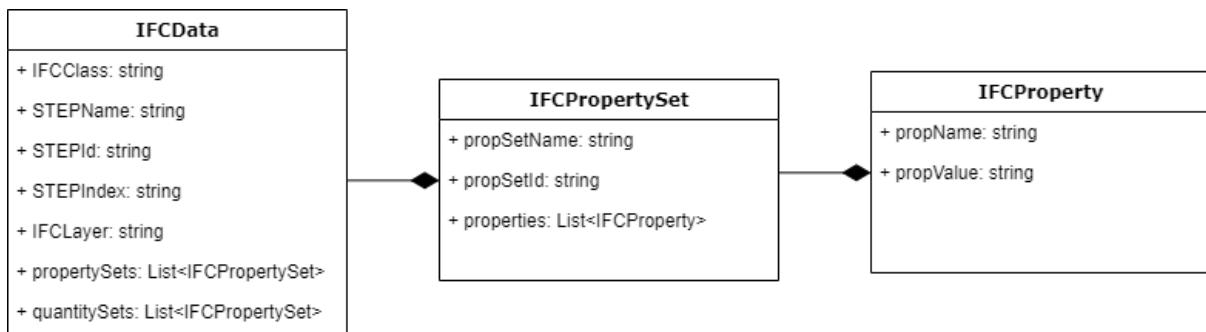
---

<sup>1</sup><https://ifcopenshell.org>

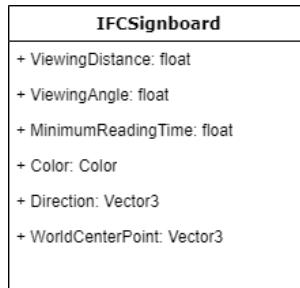
- XML: rappresenta la struttura dello schema IFC, utilizzando una gerarchia di tag XML

Dopo aver importato la geometria con i relativi materiali, **IfcOpenShellParser** esegue il parsing del file XML per associare i dati provenienti dall'*IFC Schema* alle corrispondenti parti dell’edificio. Questo passaggio permette di integrare informazioni aggiuntive, come le proprietà fisiche e funzionali delle varie componenti dell’edificio, direttamente all’interno del modello 3D.

In particolare, ad ogni elemento dell’edificio viene associata la classe **IFCData**, con le corrispondenti proprietà (Figura 4.2a).



(a) Classi utilizzate per immagazzinare i dati IFC standard.



(b) Classe utilizzata per immagazzinare i dati esclusivi alla segnaletica.

Figura 4.2: Diagramma UML delle classi usate per immagazzinare i dati provenienti dall’*IFC Schema*.

Le informazioni relative alla segnaletica possono essere integrate nel modello IFC attraverso l’uso di software BIM, seguendo i nuovi tag IFC introdotti nel Capitolo 2.2.

Durante l’importazione, questi vengono riconosciuti e associati agli elementi della segnaletica nel modello, attraverso la classe **IFCSignboard** (Figura 4.2b).

Durante la scansione del file XML si vanno anche a definire quali sono le zone percorribili dagli agenti, come *IfcSlab* (pavimenti) o *IfcStair* (scale), e le zone che gli agenti potranno attraversare liberamente, ad esempio *IfcDoor* (porte). Questo sarà fondamentale

per il funzionamento dei moduli descritti nei capitoli successivi.

Poiché lo standard IFC è estremamente flessibile e i modelli BIM possono variare notevolmente, il modulo **IfcOpenShellParser** offre una serie di parametri configurabili per adattare il processo di importazione alle esigenze specifiche di ciascun progetto (Figura 4.3).

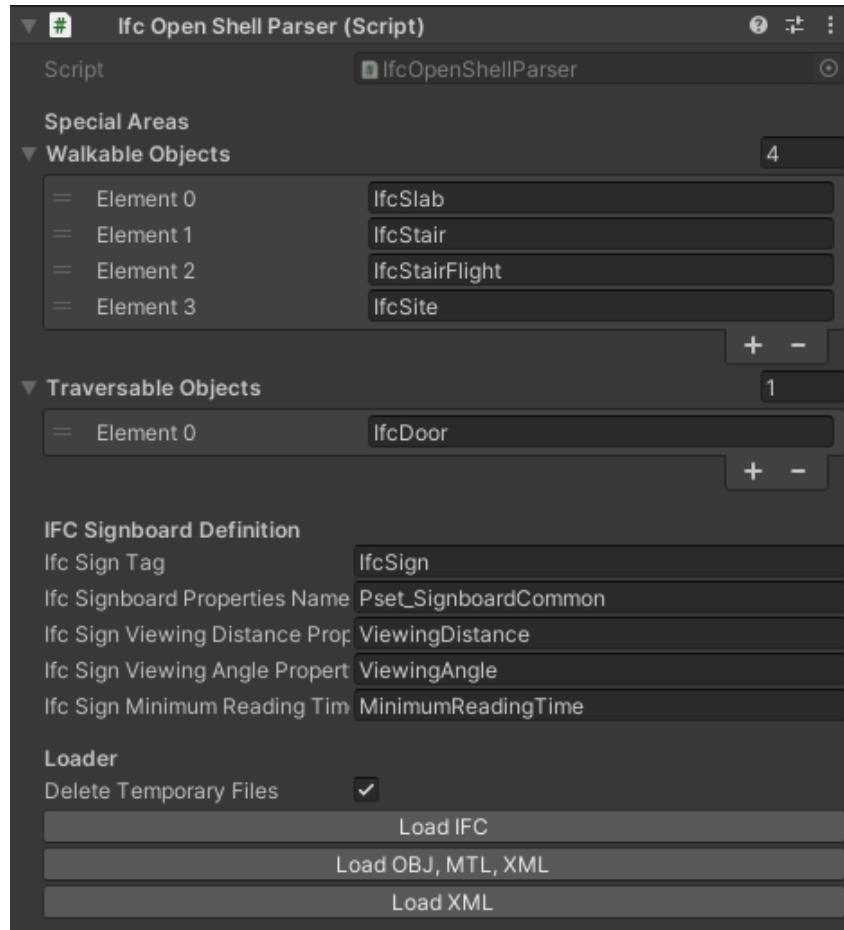


Figura 4.3: Interfaccia del modulo IfcOpenShellParser

### 4.3 Aree navigabili

Il sistema di *Navigation Mesh* (NavMesh) implementato da *Unity*, introdotto nel Capitolo 3.2.1, rappresenta uno strumento versatile e intuitivo per definire le aree percorribili dagli agenti virtuali.

Questo sistema è progettato per automatizzare il processo di calcolo delle superfici navigabili all'interno di un ambiente tridimensionale, facilitando così la gestione del movimento degli agenti.

Una delle caratteristiche chiave della NavMesh è la possibilità di generare la *NavMesh Surface*, una rappresentazione delle aree accessibili all'interno dell'edificio, tramite un processo chiamato *baking*.

Il *baking* consiste nell'analizzare il modello 3D e calcolare in anticipo le aree percorribili, memorizzando queste informazioni per un utilizzo successivo durante la simulazione. Questo processo è completamente automatico: l'utente deve solo indicare il modello da analizzare e definire le aree percorribili dagli agenti. La NavMesh Surface, evidenziata in azzurro nella Figura 4.4, rappresenta l'area su cui gli agenti possono muoversi.

Nel nostro caso andremo a confinare il movimento degli agenti alle aree delimitate dai pavimenti e dalle scale.

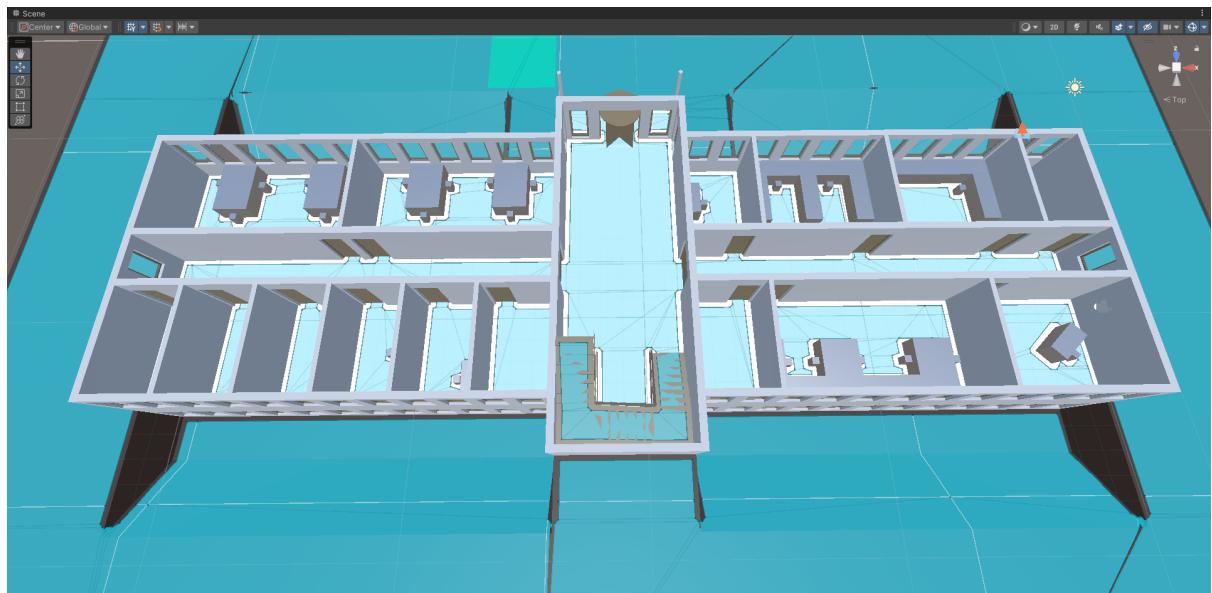


Figura 4.4: Come si presenta la NavMesh (in azzurro) di una sezione di un edificio.

Due fattori principali influenzano l'area coperta dalla NavMesh Surface: i **filtri** e i **parametri dell'agente**. Attraverso i filtri, è possibile escludere alcune parti del modello (come muri, mobili o altre strutture fisse) durante il processo di baking, definendo con precisione le aree dove gli agenti possono o non possono muoversi. Questo approccio offre un controllo granulare sulla configurazione dell'ambiente navigabile.

I **parametri dell'agente** (Figura 4.5) (altezza, larghezza e capacità di scalata o *step height/slope*) giocano un ruolo essenziale nel determinare come gli agenti interagiscono con l'ambiente circostante.

In base alle dimensioni specificate per gli agenti, la NavMesh Surface si adatta per evitare che gli agenti si trovino troppo vicini a ostacoli o attraversino spazi troppo stretti.

Nella Figura 4.4, è visibile come la superficie percorribile non si estenda fino a toccare i muri o gli elementi di arredo. Questo spazio è calcolato automaticamente in base al volume degli agenti, che impedisce loro di attraversare aree dove il loro movimento potrebbe risultare impossibile o innaturale.



Figura 4.5: Parametri del *NavMesh Agent*.

Un aspetto interessante del sistema NavMesh di Unity è che, una volta definita l'area percorribile e introdotto un oggetto con il componente NavMesh Agent, è possibile controllare il movimento dell'agente semplicemente specificando una destinazione all'interno della NavMesh Surface. Unity gestisce automaticamente il calcolo del percorso più breve per l'agente, sfruttando l'algoritmo A\* [8]. Questo è uno degli algoritmi di pathfinding più noti ed efficienti, in grado di trovare percorsi ottimali tenendo conto di eventuali ostacoli lungo il tragitto. L'agente seguirà quindi il percorso calcolato fino a raggiungere la destinazione desiderata, aggiornando dinamicamente il movimento se le condizioni dell'ambiente cambiano.

Anche se questo lavoro non farà uso degli agenti messi a disposizione da Unity nella sua forma più pura, per via delle limitazioni discusse nel Capitolo 3.2.1, l'implementazione di NavMesh sarà comunque fondamentale per guidare gli agenti all'interno dell'edificio.

## 4.4 Piani di Visibilità e VCA

È necessario ora costruire un implementazione che permetta agli agenti di individuare la segnaletica all'interno dell'ambiente.

Un approccio “*naive*” consiste nel far interagire ogni singolo agente con tutti i cartelli segnaletici circostanti a intervalli regolari per verificarne la visibilità. Sebbene efficace, come dimostrato da Felix Ruzzoli [3], questo metodo può essere ottimizzato.

Una soluzione più efficiente consiste nel definire un'area di visibilità per ciascun cartello (Figura 4.7, in blu) e salvare questa informazione in una cache. Questo approccio riduce notevolmente i costi computazionali, poiché la maggior parte dei calcoli onerosi viene eseguita una sola volta, prima dell'inizio della simulazione.

Iniziamo identificando le superfici piane percorribili dagli agenti, che nel contesto del modello BIM corrispondono alla ricerca di tutte le geometrie contrassegnate come percorribili durante la fase di import del modello (Capitolo 4.2).

Utilizzando le impostazioni di default queste corrispondono agli elementi marchiati come *IfcSlab*, ossia i pavimenti.

Nel modello tridimensionale queste geometrie vengono rappresentate attraverso l'utilizzo di *mesh*, ovvero un insieme di vertici, raggruppati in triangoli che definiscono un'area e/o un volume. (Figura 4.6).

Da ciascuna di queste mesh, isoliamo la superficie superiore, che andremo a chiamare **Piano di Visibilità**.

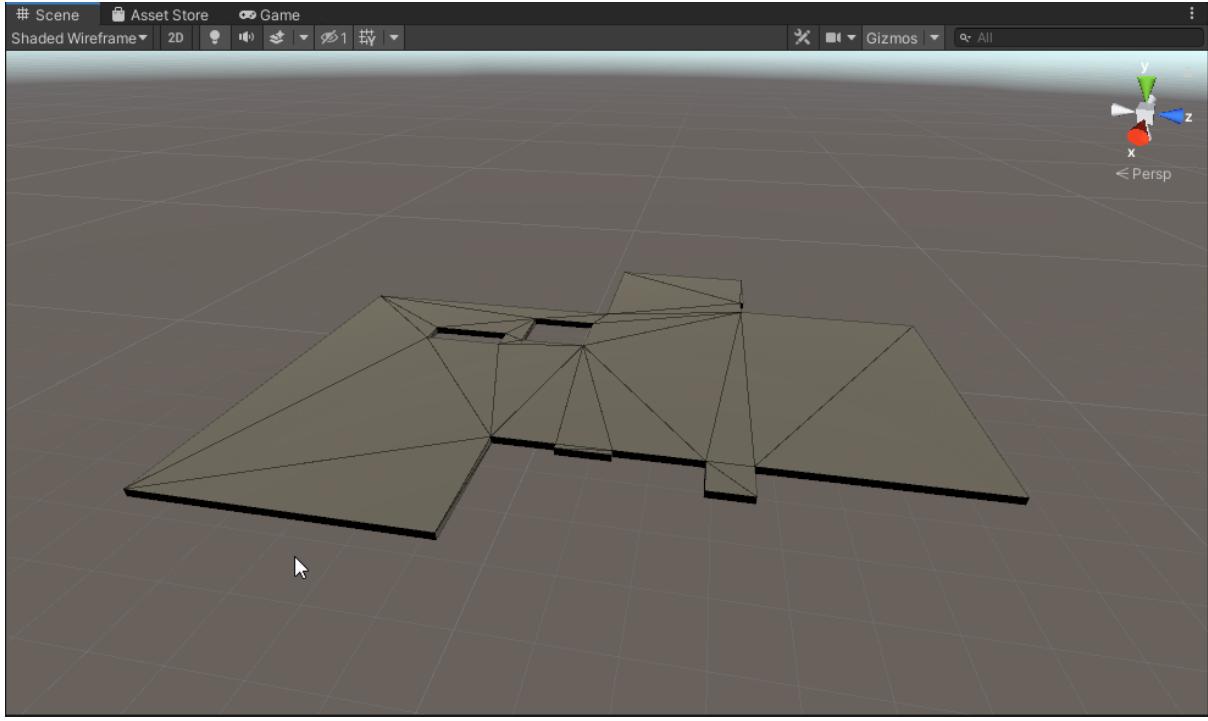


Figura 4.6: Esempio di *IfcSlab* con evidenziati i triangoli che compongono la mesh

L'**area di visibilità** (o **VCA**) di ogni cartello può essere definita come parte di questo piano.

Secondo quanto descritto da Motamed et al. [15], essa è l'intersezione di una sezione sferica e un piano:

$$X = V \cap G \quad (4.1)$$

Dove:

- $V$  è un volume conico, descritto dall'Equazione 4.2, dove  $p$  e  $n$  indicano rispettivamente il punto centrale e la direzione del cartello (rappresentata dal vettore normale alla superficie del cartello):

$$V = \left\{ v_i \mid \frac{(v_i - p) \cdot n}{\|v_i - p\| \|n\|} \geq \cos\left(\frac{\theta}{2}\right) \quad \text{e} \quad \|v_i - p\| \geq d \right\} \quad (4.2)$$

- $G$  è un piano parallelo al suolo, posizionato all'altezza degli occhi dell'agente.

I parametri  $d$  e  $\theta$  rappresentano rispettivamente la distanza massima da cui il cartello è leggibile e l'angolo massimo da cui è visibile. Entrambi i parametri vengono definiti dall'utente, in base a variabili come, ad esempio, la dimensione del testo sul cartello [15].

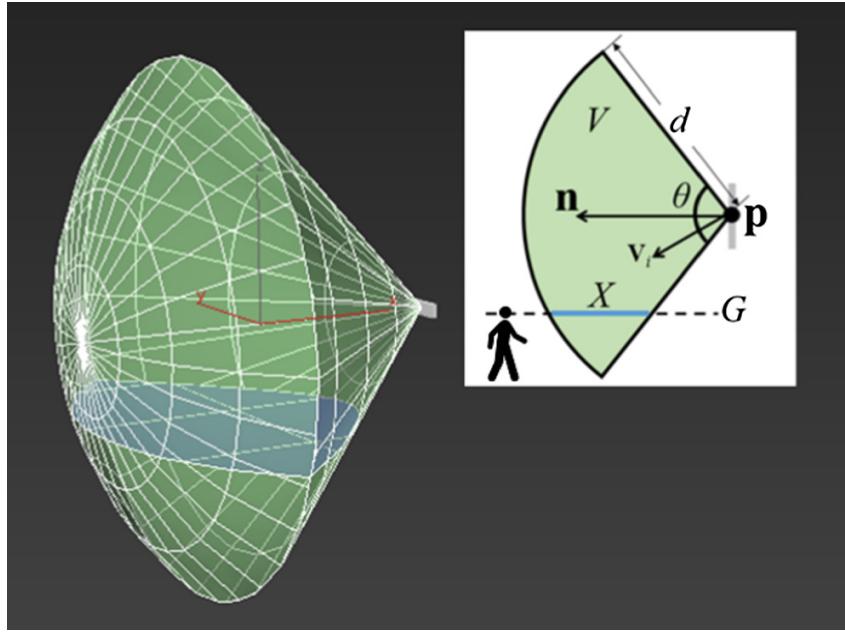


Figura 4.7: Area di visibilità di un cartello. Fonte: [15]

Per determinare  $G$ , il **Piano di Visibilità** viene elevato all'altezza degli occhi dell'agente. Questo approccio rende semplice supportare diverse tipologie di agenti, regolando il Piano di Visibilità a diverse altezze.

Ad esempio, per agenti come adulti e persone su sedia a rotelle, il sistema calcolerà due insiemi distinti di aree di visibilità, corrispondenti alle diverse altezze.

Infine, è necessario determinare il punto  $v_i$ , che, durante un ipotetico calcolo in tempo reale, corrisponderebbe alla posizione dell'agente. Poiché il calcolo viene eseguito a priori, adottiamo un approccio diverso: per ogni **Piano di Visibilità**, creiamo una griglia di punti equidistanti, con una risoluzione variabile, che copre l'intera superficie del piano (Figura 4.8).

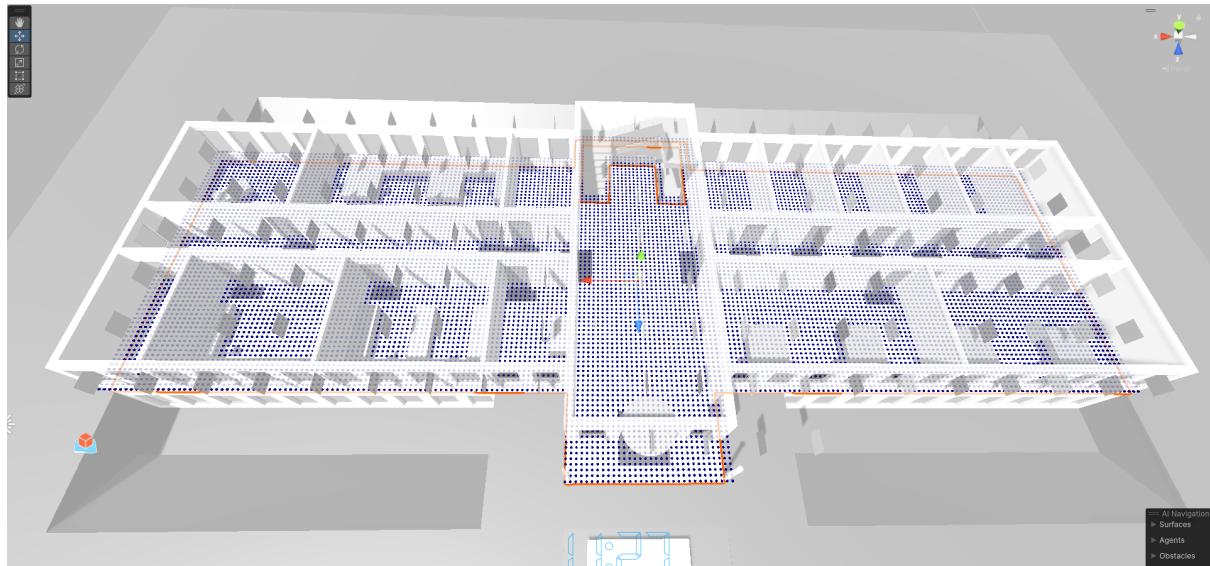


Figura 4.8: Un *Visibility Plane* con evidenziati in blu i punti che compongono la griglia, con una risoluzione di 5 punti/metro

Per ogni cartello, ognuno di questi punti deve soddisfare due condizioni per essere considerato parte della sua *VCA* (Figura 4.9):

1. *Deve soddisfare le disequazioni di Equazione 4.2*, utilizzando i parametri intrinseci del cartello in questione.
2. *Non deve esserci alcun ostacolo tra il punto  $p$ , il centro del cartello ed il punto  $v_i$  del piano, che stiamo considerando*: questo viene verificato usando la funzione *Raycast* del motore fisico di Unity. Viene lanciato un raggio da un punto verso una direzione, definita da un vettore, (dal punto  $p$  in direzione del punto  $v_i$ ) e si verifica che il raggio non abbia incontrato alcun ostacolo.

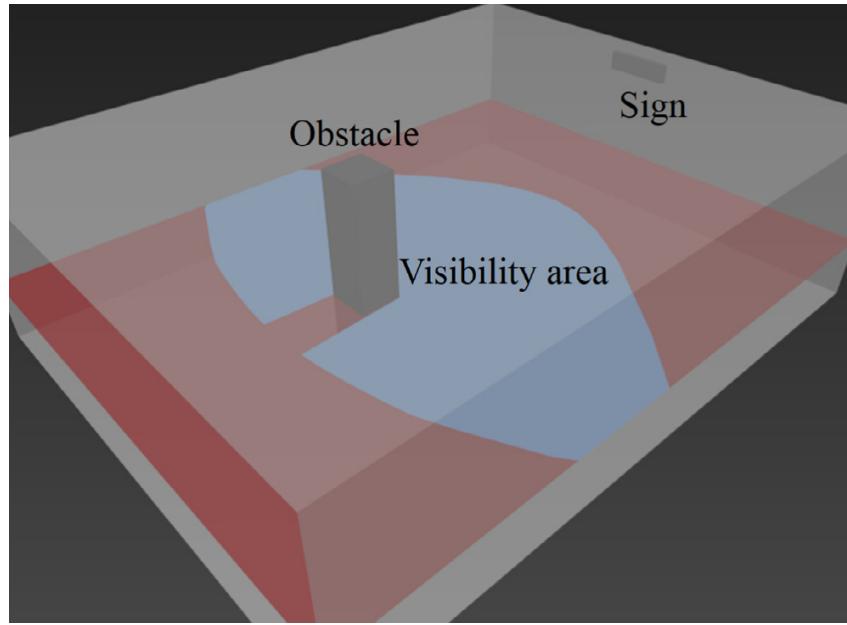


Figura 4.9: Esempio di *area di visibilità* (blu), evidenziata all'interno di un *piano di visibilità* (rosso). Fonte: [15]

Per ciascun **Piano di Visibilità** viene creata una matrice sparsa, in cui ogni cella corrisponde ad un punto della griglia. Se un punto soddisfa tutti i requisiti, le sue coordinate vengono inserite all'interno di essa, specificando a quale cartello appartiene.

Questi dati possono essere utilizzati successivamente durante la simulazione per verificare se un agente si trova all'interno di una *VCA*, semplicemente interrogando la cella della griglia in cui l'agente si trova.

#### 4.4.1 Visibility Handler

Il concetti introdotti nel capitolo precedente vengono implementati dal modulo **Visibility Handler**, che si occupa di generare ed immagazzinare tutte le informazioni relative alla visibilità dei cartelli.

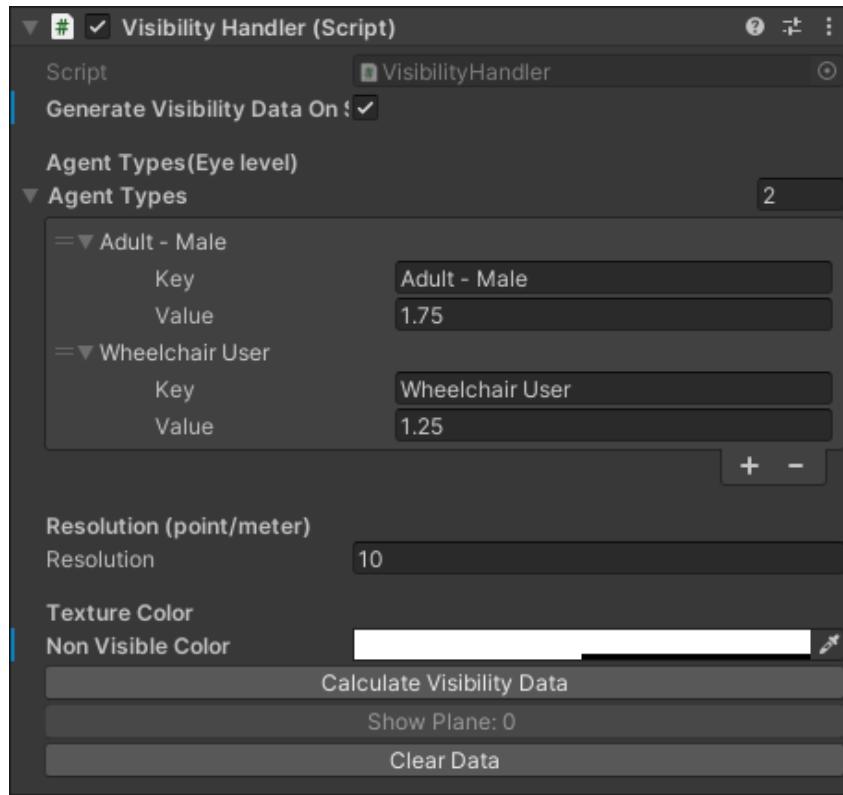


Figura 4.10: Interfaccia del modulo Visibility Handler

Attraverso la sua interfaccia (Figura 4.10) è possibile specificare:

- Se generare le informazioni relative alle VCA immediatamente appena si entra in fase di esecuzione (fase di *Start*)
- I tipi di agente di cui si vuole tenere conto durante la simulazione
- La risoluzione della griglia introdotta nel capitolo precedente
- Il colore del piano di visibilità in un punto in cui non è visibile alcun cartello

Esso è accompagnato dal modulo **Visibility Plane Generator** che si occupa esclusivamente di generare i *Piani di Visibilità*, dando la possibilità all'utente di specificare ulteriori *tag IFC* da utilizzare come base.

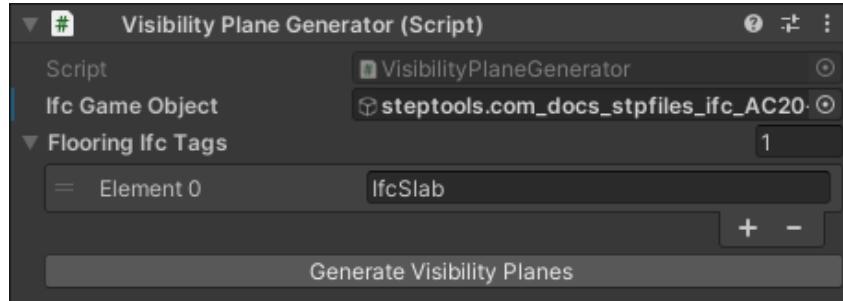


Figura 4.11: Interfaccia del modulo Visibility Plane Generator

Una volta generati i dati, viene mostrato il *Visibility Plane* relativo al primo tipo di agente della lista, con evidenziate su di esso le VCA dei cartelli presenti (Figura 4.12). È possibile visualizzare gli altri *Piani di Visibilità* attraverso il pulsante *Show Plane*.

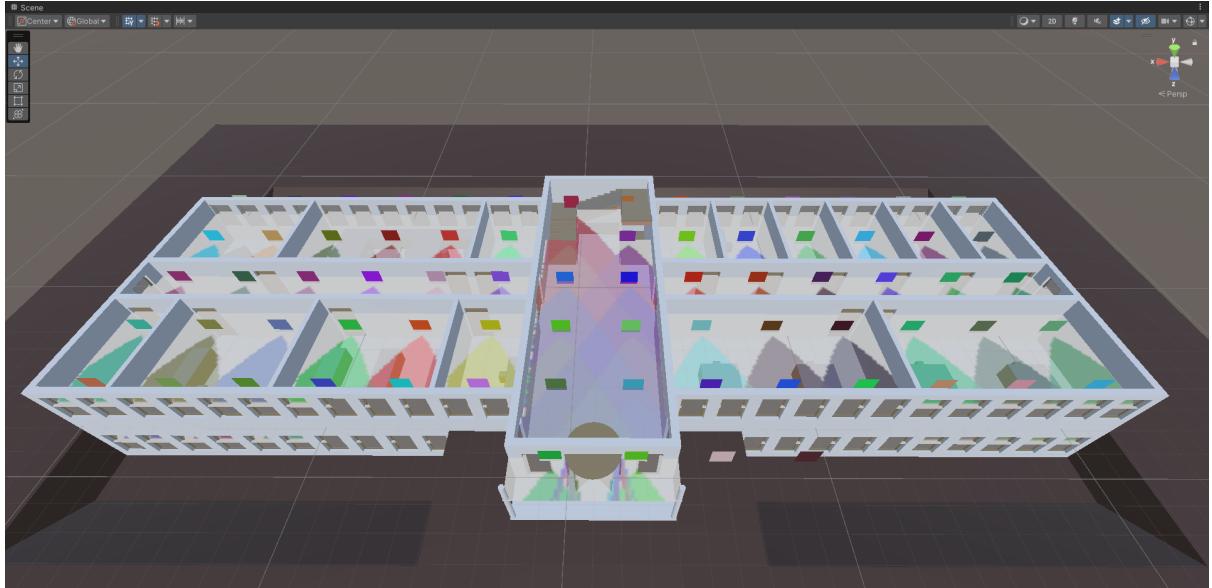


Figura 4.12: Piano di Visibilità del primo piano di un edificio. Da notare come il piano è sollevato all'altezza degli occhi di un agente (1.75m) ed i colori delle VCA corrispondono ai colori dei cartelli.

## 4.5 Generazione di segnaletica

Oltre a poter analizzare la visibilità della segnaletica già presente nel modello BIM importato, può essere utile analizzare un insieme di cartelli che vanno a coprire l'intera area percorribile dagli agenti.

Il modulo **Signboard Grid Generator** è stato sviluppato esattamente per questo scopo. Questo strumento è particolarmente utile in fase di analisi, poiché permette di identificare le aree dell'edificio che si prestano meglio all'affissione della segnaletica e di confrontare diverse configurazioni di cartelli.

La segnaletica viene generata automaticamente, posizionando i cartelli sui vertici di una griglia regolare, equidistanti sugli assi orizzontali. La distanza tra i cartelli può essere regolata attraverso il parametro "Resolution" (Figura 4.13), che è espresso in punti per metro. Questo parametro permette di controllare la densità della griglia, consentendo di variare la risoluzione in base alle dimensioni dell'edificio o all'accuratezza dell'analisi richiesta.

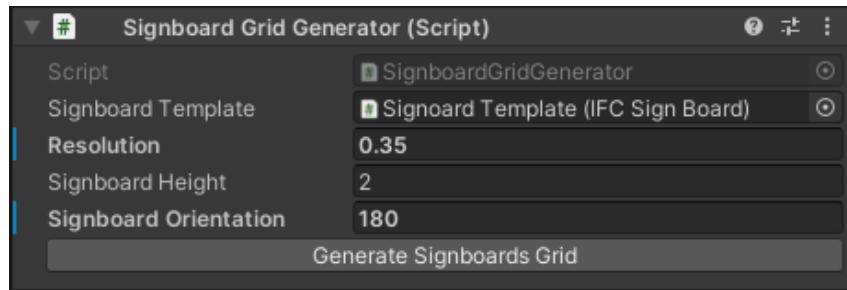


Figura 4.13: Interfaccia del modulo Signboard Grid Generator

Oltre alla risoluzione, è possibile specificare ulteriori parametri, come l'**altezza** (in metri) dei cartelli e il loro **orientamento orizzontale** (in gradi), ovvero la rotazione intorno all'asse verticale. L'altezza dei cartelli è particolarmente importante, poiché deve essere adattata in base alla tipologia di agente che si desidera simulare, come ad esempio adulti, bambini o persone su sedia a rotelle. L'orientamento dei cartelli può essere regolato per ottimizzare la loro visibilità in funzione delle caratteristiche dell'ambiente, come la disposizione dei corridoi o la posizione delle porte.

Questo strumento consente quindi di testare diverse configurazioni di segnaletica, in modo da poterle mettere a confronto oppure osservare in quali zone gli utenti interagiscono più spesso con la segnaletica.

In Figura 4.12 è possibile osservare un esempio di griglia di cartelli generata da questo modulo, con i parametri mostrati in Figura 4.13.

## 4.6 Marker e Routing

Per garantire che gli agenti si muovano con un certo livello di realismo all'interno dell'edificio e interagiscano efficacemente con la segnaletica, è necessario fornire loro punti di riferimento che fungano da destinazioni o nodi di decisione durante la navigazione. Permettere agli agenti di muoversi in modo erratico non fornirebbe risultati affidabili né garantirebbe che tutta la segnaletica abbia uguali possibilità di essere notata dagli agenti. Per affrontare questo problema, sono stati sviluppati dei moduli per identificare una rete di **marker** o **waypoint** all'interno dell'edificio.

Dopo aver osservato diversi modelli di edifici open-source, si è ipotizzato che l'uso delle porte e scale (elementi contrassegnati con il tag *IfcDoor* o *IfcStair* nel modello IFC) come marker sia sufficiente fornire agli agenti la possibilità coprire gran parte della superficie di diversi edifici.

Questo perché, prendendo una coppia casuale di punti A e B all'interno di un edificio, a

una distanza tale che sia necessario l’uso della segnaletica, è quasi garantito che l’agente debba attraversare diversi marker lungo il percorso. Inoltre, la distanza tra l’ultimo marker e la destinazione sarà relativamente piccola rispetto al percorso totale.

I marker facilitano la navigazione all’interno dell’edificio poiché riducono il numero di decisioni che un agente deve prendere durante il suo movimento: invece di muoversi liberamente e in modo disordinato, l’agente può semplicemente selezionare uno dei marker visibili nelle vicinanze e dirigersi verso di esso.

Possiamo immaginare i marker come nodi di un grafo, collegati da archi che rappresentano percorsi diretti tra marker. Un arco collega due marker solo se questi possono essere raggiunti l’uno dall’altro senza attraversare un altro marker intermedio. La presenza di marker semplifica sostanzialmente la creazione di un “*global planner*”[12], un termine preso in prestito dalla robotica che indica un sistema che utilizza informazioni a priori per calcolare un percorso ottimale. Il global planner può quindi inviare all’agente una sequenza di archi, ossia di percorsi tra marker, che l’agente seguirà per coprire il percorso completo.

In questo contesto, il *local planner* sarà gestito dal sistema NavMesh di Unity, che si occuperà di calcolare il percorso esatto tra un marker e l’altro, garantendo che l’agente segua un tragitto efficiente attraverso l’edificio.

**Tipologie di Marker** Esistono due principali tipologie di marker che vengono utilizzate nel sistema:

- **Intermediate Marker:** Questo è il tipo base di marker, posizionato nei punti principali dell’ambiente (come le porte). Viene utilizzato per guidare gli agenti nelle aree chiave dell’edificio, fornendo punti di riferimento che l’agente può utilizzare per decidere la propria destinazione.
- **Linked Marker:** Questo tipo di marker è particolarmente utile in casi specifici, come le scale o altre aree in cui il percorso è obbligato. Un Linked Marker può essere collegato ad un altro marker, comunicando all’agente che, una volta raggiunto, deve proseguire verso il marker collegato. Questa caratteristica permette agli agenti di muoversi su più piani in modo autonomo. Ad esempio, quando un agente raggiunge un *Linked Marker* posto all’ingresso di una scala, viene obbligato a proseguire verso il marker collegato posto all’uscita della scala, garantendo una corretta navigazione tra i diversi piani dell’edificio.

Questo sistema di marker, combinato con un routing efficace, consente di simulare il movimento degli agenti in modo strutturato e realistico all'interno dell'edificio, offrendo una soluzione efficiente per la navigazione in spazi tridimensionali complessi.

#### 4.6.1 Marker Generator

*Marker Generator* è il modulo responsabile della scansione degli elementi nel modello BIM, alla ricerca dei punti più adatti per posizionare i marker introdotti nel capitolo precedente. Di default, questo modulo posiziona un *Intermediate Marker* alla base di ogni elemento con tag *IfcDoor*, e un *Linked Marker* all'inizio e alla fine di ogni rampa di scale (elementi contrassegnati con il tag *IfcStair* o *IfcStairFlight*), ma è possibile specificare ulteriori tag per personalizzare il posizionamento dei marker in base alle esigenze del progetto.

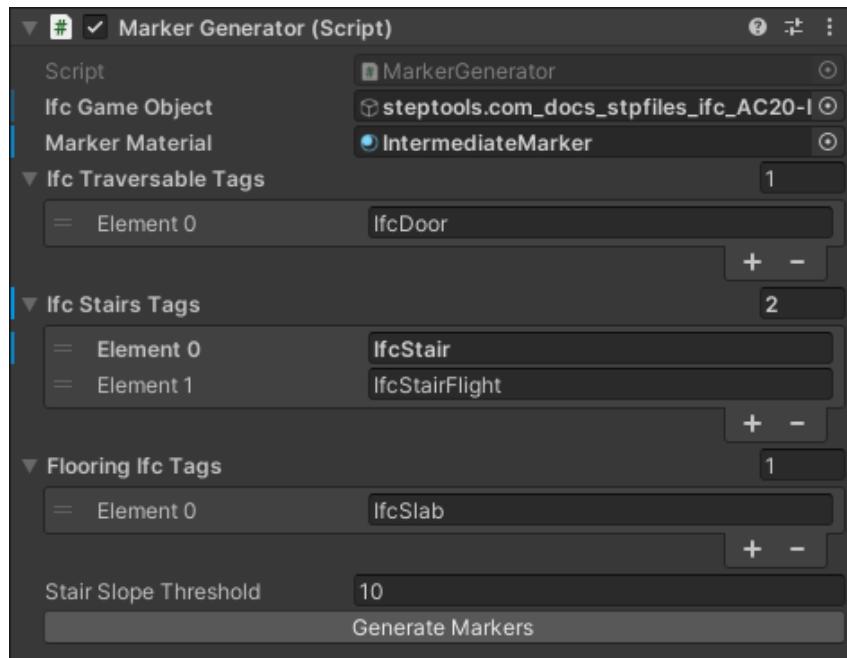


Figura 4.14: Interfaccia del modulo Marker Generator.

La generazione degli *Intermediate Marker* è un processo relativamente semplice: una volta identificato un elemento con il tag corretto, viene posizionato un marker alla sua base. Le dimensioni del marker sono adattate in base alla sezione orizzontale dell'elemento, garantendo un allineamento accurato con le dimensioni fisiche dell'oggetto.

Il processo per generare i *Linked Marker*, invece, è più complesso. Dopo aver individuato una scala, il modulo esamina la forma della *NavMesh Surface* (Capitolo 4.3) nell'area in cui la interseca. Se viene individuato una rampa con un dislivello superiore a

un valore di soglia definito dal parametro “*Stair Slope Threshold*”, questa viene considerata un candidato per il posizionamento del marker.

Successivamente, vengono eliminati i marker non allineati con i piani dell’edificio, per evitare posizionamenti inutili sui pianerottoli.

Se si osserva attentamente la Figura 4.15, si può osservare come il marker (in blu) viene posizionato in concomitanza dell’inizio della rampa sulla *NavMesh Surface* (in azzurro).

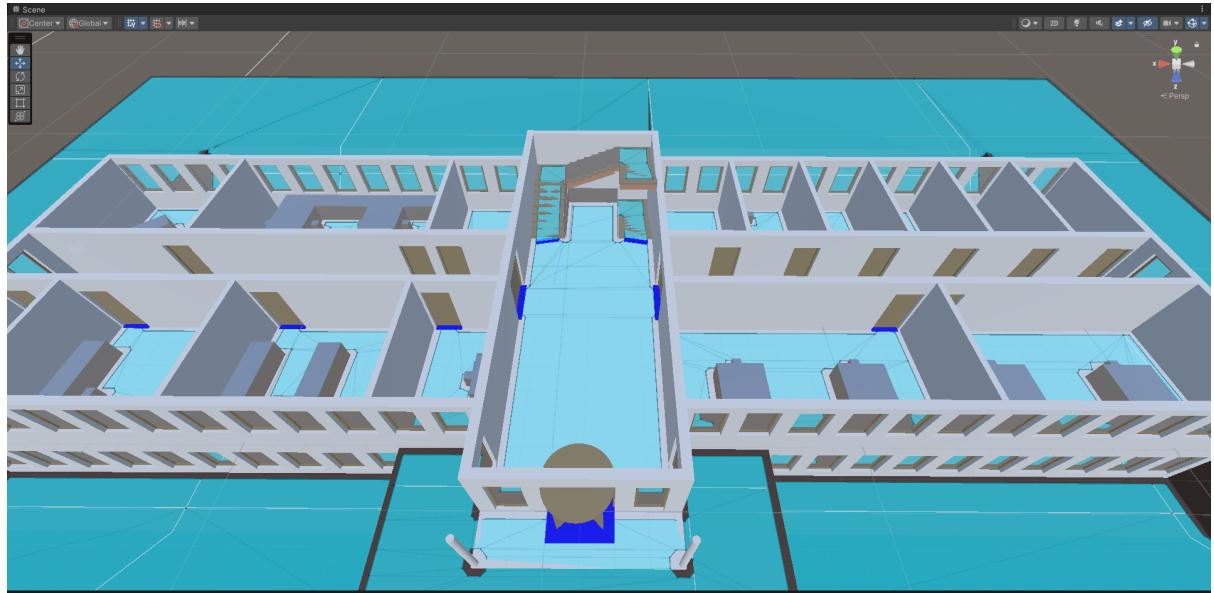


Figura 4.15: Esempio di marker (in blu) in una sezione di un edificio.

Infine, i *Linked Marker* individuati vengono collegati tra di loro, seguendo l’ordine di altezza. Questo collegamento garantisce che, quando un agente entra in una rampa di scale, possa essere guidato correttamente fino alla fine della stessa, consentendogli di spostarsi autonomamente tra i piani.

# Capitolo 5

## Simulazione del flusso pedonale

### 5.1 Agenti e ambiente

L’architettura dei diversi tipi di agente che andremo a presentare si compone di diversi moduli e per ognuno di essi bisognerebbe fare delle considerazioni separate, ma volendo trattare le sue componenti come una sola, essi rientrano nella definizione di **agenti istoretici**, ovvero dotati di uno stato interno. Questo stato permette agli agenti di conservare informazioni sul loro comportamento precedente e utilizzarle per prendere decisioni future.

Gli agenti conoscono la propria posizione ed orientamento nello spazio e sono in grado di prendere decisioni osservando l’area circostante.

Questo rende l’ambiente in cui si trovano **inaccessibile**, il che significa che non possono accedere all’intera configurazione dell’ambiente in ogni momento. Gli agenti, infatti, agiscono in base a informazioni locali e aggiornano costantemente la loro percezione man mano che si muovono nello spazio.

L’ambiente può essere descritto come **statico**, **continuo** e **sequenziale**:

- **Statico:** una volta importato il modello BIM ed inserito la segnaletica, l’ambiente non subisce modifiche nel tempo. L’unica parte dinamica sono gli agenti, che aggiornano costantemente la loro posizione e possono interagire tra loro ostacolandosi lungo i percorsi.
- **Continuo:** sebbene in alcuni casi lo spazio venga discretizzato per migliorare le prestazioni, gli agenti non occupano posizioni predefinite. Al contrario, si muovono liberamente nello spazio tridimensionale.
- **Sequenziale:** Gli agenti prendono decisioni in ogni momento, a ogni aggiornamento della simulazione. Questo significa che il loro comportamento evolve nel tempo in

base alle condizioni locali e alle interazioni con l’ambiente.

Un altro aspetto rilevante è la gestione delle collisioni all’interno dell’ambiente. Il motore fisico di Unity, utilizzato per rilevare le collisioni tra gli elementi della scena, è deterministico, a meno di piccoli errori legati alla gestione dei numeri in virgola mobile. Di conseguenza, possiamo descrivere l’ambiente stesso come **deterministico**, nel senso che dati gli stessi input, il sistema produrrà sempre lo stesso risultato, garantendo una simulazione consistente e ripetibile nella maggior parte dei casi.

## 5.2 Architettura degli agenti

Gli agenti utilizzati nella simulazione posso fare uso di diverse componenti modulari, non necessariamente mutualmente esclusive. Questo approccio permette di combinare facilmente vari elementi, consentendo l’estensione delle funzionalità o la creazione di agenti con comportamenti personalizzati in base agli obiettivi della simulazione.

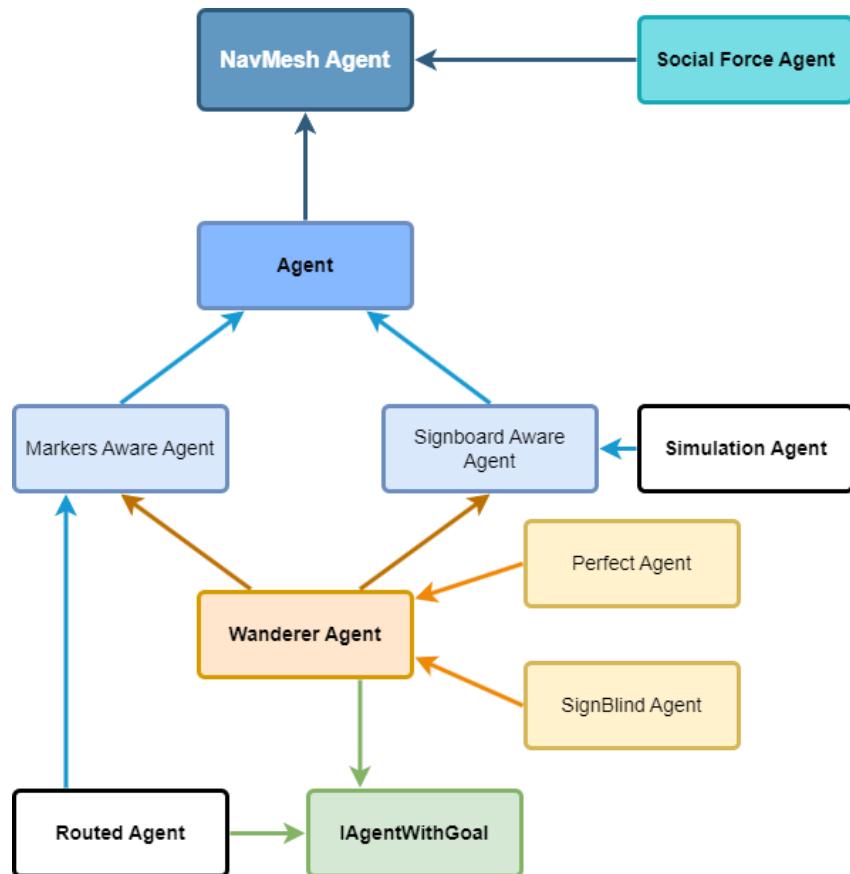


Figura 5.1: Diagramma delle dipendenze tra i vari componenti degli agenti.

In Figura 5.1 è riportato il diagramma che illustra i moduli sviluppati e le relazioni di dipendenza tra di essi. Le frecce indicano la direzione della dipendenza, ovvero da un

modulo dipendente al modulo da cui esso dipende.

I moduli possono essere combinati e applicati a oggetti nella scena, offrendo grande flessibilità nella costruzione di agenti personalizzati per diversi scenari di simulazione. Questa architettura modulare promuove il riutilizzo delle componenti e rende possibile l'introduzione di nuove funzionalità senza la necessità di riscrivere interamente il codice, mantenendo la struttura organizzata e scalabile.

### 5.2.1 NavMesh Agent

Il **NavMesh Agent** è un componente incluso tra gli strumenti forniti da Unity per la gestione della navigazione. È una delle parti fondamentali del sistema NavMesh, ed è necessario per designare un oggetto come agente all'interno di un'area percorribile (Capitolo 4.3), abilitando così il movimento all'interno della *NavMesh Surface*.

Esso mette a disposizione una lista estesa di funzionalità, ma dati i requisiti di design, ne varrà utilizzato un insieme ridotto.

**NavMesh Agent** agirà principalmente come *local planner* [12], ovvero sarà responsabile della generazione di una serie di waypoint ogni volta che viene impostata una nuova destinazione. Questo processo tiene conto degli ostacoli presenti lungo il tragitto e genera percorsi ottimizzati per brevi distanze, garantendo che l'agente segua un percorso fluido e coerente all'interno dell'ambiente.

### 5.2.2 Social Force Agent

Il modulo **Social Force Agent** si sostituisce all'implementazione del movimento dell'agente da parte di **NavMesh Agent**, ricevendo da quest'ultimo solo informazioni sulla posizione del waypoint immediatamente successivo e velocità desiderata.

Esso implementa le forze sociali introdotte nel Capitolo 3.2.2, basandosi sul seguente modello matematico:

$$\frac{dv_i}{dt} = f_i^0 + f_i^{\text{wall}} + f_{ij}$$

In breve, il movimento di un agente nell'ambiente è determinato dalla somma di tre forze:

- **Driving force** ( $f_i^0$ ): rappresenta lo stimolo del pedone di muoversi in una direzione specifica a una determinata velocità. Manipolando questa forza, è possibile guidare il pedone lungo la traiettoria desiderata.
- **Forza di repulsione dai muri** ( $f_i^{\text{wall}}$ ): allontana il pedone dalle pareti, evitando collisioni con gli ostacoli fissi.

- Forza di interazione tra agenti ( $f_{ij}$ ): rappresenta l'interazione tra due pedoni  $i$  e  $j$ , con una forza repulsiva di intensità inversamente proporzionale alla loro distanza.

$f_i^0$  e  $f_{ij}$  sono state implementate seguendo esattamente il modello descritto in “*Experimental study of the behavioural mechanisms underlying self-organization in human crowds*” [16], usando gli stessi parametri risultanti dai loro **test empirici**.

**Driving Force** L’implementazione di  $f_i^0$  riceve il target successivo e la velocità desiderata da *NavMesh Agent* e genera un vettore direzione verso il punto da raggiungere. L’intensità del vettore è proporzionale alla velocità desiderata in quell’istante. La formula utilizzata è:

$$\vec{f}_i^0 = \frac{v_i^{\text{des}} \cdot \hat{e}_i - \vec{v}_i}{\tau}$$

Dove:

- $v_i^{\text{des}} = 1.29 \pm 0.19 \text{ m/s}$ : La velocità desiderata dell’agente  $i$ .
- $\hat{e}_i$ : Il vettore direzionale unitario che indica la direzione verso la destinazione dell’agente  $i$ .
- $\vec{v}_i$ : La velocità attuale dell’agente  $i$ .
- $\tau = 0.54$ : Il tempo di rilassamento, che rappresenta quanto velocemente l’agente corregge la sua velocità attuale per avvicinarsi a quella desiderata. Un tempo di rilassamento più breve indica un adattamento più rapido.

**Forza di interazione tra agenti** L’interazione tra una qualsiasi coppia di agenti  $i$  e  $j$  è descritta dalle seguenti equazioni:

$$\begin{aligned} \vec{f}_{ij}(d, \theta) &= -Ae^{-d/B} \left[ e^{-(n'B\theta)^2} \vec{t}_{ij} + e^{-(nB\theta)^2} \vec{n}_{ij} \right] \\ \vec{t}_{ij} &= \frac{\lambda \cdot (\vec{v}_i - \vec{v}_j) + \vec{e}_{ij}}{\|\lambda \cdot (\vec{v}_i - \vec{v}_j) + \vec{e}_{ij}\|} \\ B &= \gamma \cdot \|\lambda \cdot (\vec{v}_i - \vec{v}_j) + \vec{e}_{ij}\| \end{aligned}$$

Dove:

1.  $A$ : Parametro di intensità della forza. Modula la grandezza dell’interazione tra gli agenti.

2.  $d$ : Distanza tra l'agente  $i$  e l'agente  $j$ . Questo parametro influenza l'attenuazione della forza con la distanza.
3.  $B$ : Parametro che regola la portata dell'interazione. Viene calcolato come  $B = \gamma \cdot \|\lambda \cdot (\vec{v}_i - \vec{v}_j) + \vec{e}_{ij}\|$ , dove  $\gamma$  è una costante e  $\lambda$  modula la differenza di velocità.
4.  $\theta$ : Angolo tra la direzione dell'interazione  $\vec{t}_{ij}$  e la linea che unisce i due agenti  $i$  e  $j$ , tenendo conto del loro campo visivo.
5.  $\vec{t}_{ij}$ : Vettore tangenziale dell'interazione, che dipende dalla velocità relativa degli agenti e dalla direzione reciproca  $\vec{e}_{ij}$ .
6.  $\vec{n}_{ij}$ : Vettore normale dell'interazione, ortogonale alla direzione di interazione  $\vec{t}_{ij}$ .
7.  $\lambda$ : Parametro che modula la differenza di velocità tra gli agenti  $i$  e  $j$ .
8.  $\gamma$ : Parametro di scala che modula la portata della forza in relazione alla distanza e alla velocità relativa.
9.  $\vec{v}_i$  e  $\vec{v}_j$ : Velocità rispettive degli agenti  $i$  e  $j$ .
10.  $\vec{e}_{ij}$ : Vettore unitario che rappresenta la direzione da  $i$  verso  $j$ , cioè la linea retta che collega i due agenti.

I valori delle costanti vengono estratti dalle seguenti distribuzioni normali:

- $\lambda \sim \mathcal{N}(2.0, 0.2)$
- $\gamma \sim \mathcal{N}(0.35, 0.01)$
- $n' \sim \mathcal{N}(3.0, 0.7)$
- $n \sim \mathcal{N}(2.0, 0.1)$
- $A \sim \mathcal{N}(4.5, 0.3)$

L'implementazione originale è stata modifica in modo non particolarmente significativo per migliorare le performance della simulazione. Ad esempio, è stato introdotto un parametro che permette di ignorare le interazioni tra agenti sufficientemente distanti.

**Forza di repulsione dai muri**  $f_i^{\text{wall}}$  ha richiesto un’implementazione ad-hoc per l’ambiente di Unity, in quanto l’ambiente utilizzato da Mehdi Moussaïd et al.[16] utilizza un sistema di gestione delle pareti considerevolmente diverso.

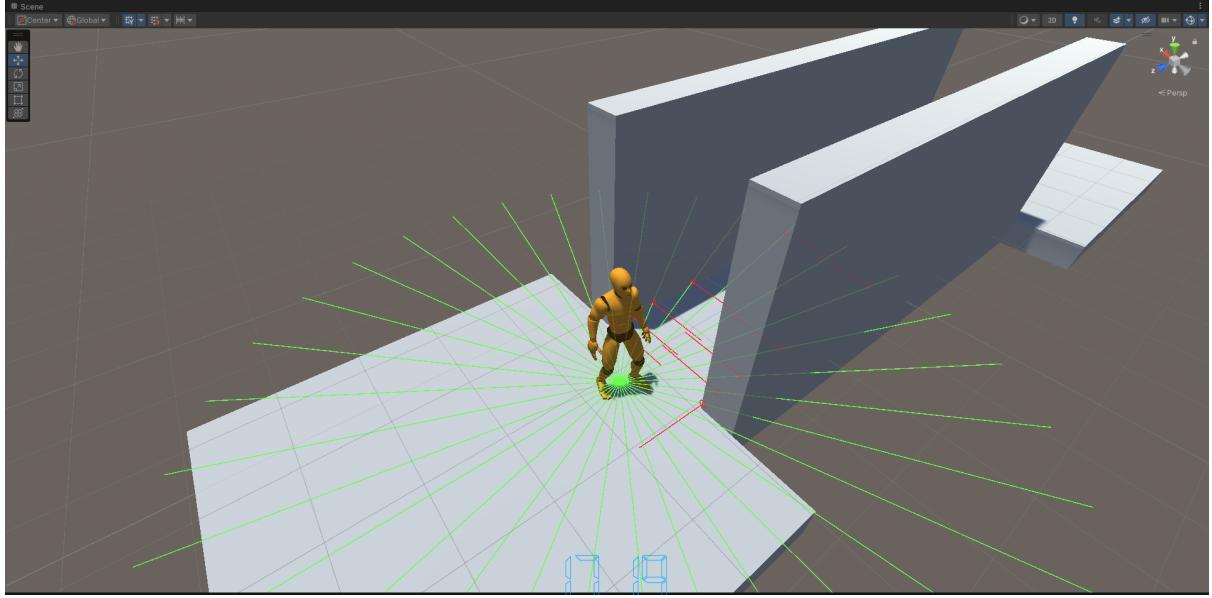


Figura 5.2: Rappresentazione grafica della funzione di Raycast. In verde i raggi che vanno alla ricerca del muro ed in rosso le collisioni.

Attraverso una funzione di **RayCast** (Figura 5.2), vengono lanciati 32 raggi da ogni agente per rilevare la distanza dai muri.

Il punto più vicino viene utilizzato per calcolare il vettore in questione:

$$f^{\text{wall}}(d_w) = a * e^{-d_w/b}$$

Dove:

1.  $a = 3$ : Coefficiente di repulsione.
2.  $b = 0.1$ : Coefficiente di decadimento.
3.  $d_w$ =distanza dal muro più vicino.

### 5.2.3 Agent

Il modulo *Agent* è componente essenziale per qualsiasi agente introdotto nell’ambiente. Le sue responsabilità sono:

- Interagire con il modulo **Agents Handler**, che tiene traccia di tutti gli agenti nella scena, gestendo la loro introduzione e rimozione dall’ambiente.

- Incapsulare le funzionalità offerte da *NavMesh Agent*, per far sì che sviluppi futuri non accedano a funzionalità che vanno in conflitto con l'attuale implementazione.
- Mettere a disposizione il valore del campo visivo dell'agente agli altri moduli
- Visualizzare informazioni di debug
- Gestione del modello 3D dell'agente

#### 5.2.4 Markers Aware Agent

Il modulo **Markers Aware Agent** permette alle altre componenti di ricevere notifiche ogni volta che un agente attraversa un marker, facilitando la navigazione all'interno dell'edificio.

Questo modulo monitora costantemente le collisioni con i marker presenti nella scena e, quando si verifica una collisione, attiva l'evento seguente, a cui altri moduli possono sottoscriversi<sup>1</sup>:

```
1 public delegate void OnMarkerReached(IRouteMarker marker);
2 public event OnMarkerReached MarkerReachedEvent;
```

L'uso di un sistema di eventi permette di mantenere un'architettura modulare e pulita. `updateFrequencyHz` è completamente indipendente dai moduli che sfruttano le sue funzionalità, garantendo flessibilità e separazione delle responsabilità all'interno del sistema.

Oltre alla gestione delle collisioni, **Markers Aware Agent** offre anche la possibilità di ottenere informazioni sui marker circostanti un agente, attraverso il seguente metodo:

```
1 public List<IRouteMarker> GetMarkersAround(float maxDistance ,
    float minDistance)
```

Questo metodo restituisce una lista di marker presenti attorno all'agente che lo chiama, filtrati in base alle distanze minima e massima specificate. Questa funzionalità è utile per prendere decisioni di navigazione basate sui marker visibili e vicini.

#### 5.2.5 Signboard Aware Agent

**Signboard Aware Agent** permette agli agenti di essere costantemente consapevoli della segnaletica circostante. Grazie al parametro `updateFrequencyHz`, che

---

<sup>1</sup><https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-subscribe-to-and-unsubscribe-from-events>

regola il ritmo degli aggiornamenti, l'agente controlla regolarmente quali cartelli segnaletici sono visibili dalla sua posizione attuale.

Questo processo mantiene aggiornata una lista di cartelli `IFCSignboard` visibili, che può essere consultata da altri moduli in qualsiasi momento.

Analogamente a *Markers Aware Agent*, questo modulo mette a disposizione l'evento **OnAgentEnterVisibilityArea**, che viene attivato non appena l'agente entra in una nuova *VCA* e l'evento **OnAgentExitVisibilityArea** quando esce.

```
1 public delegate void OnAgentInVisibilityArea(List<
    IFCSignBoard> visibleBoards, IFCSignBoard signboard, int
    agentTypeID);
2 public event OnAgentInVisibilityArea
    OnAgentEnterVisibilityArea;
3 public event OnAgentInVisibilityArea
    OnAgentExitVisibilityArea;
```

Una volta attivato, l'evento di entrata non viene nuovamente generato per la stessa *VCA*, finché l'agente non esce dall'area e rientra in seguito, evitando notifiche ridondanti.

È importante notare che, durante tutte le sue funzioni, **Signboard Aware Agent** filtra tutti i cartelli che non sono nel campo visivo dell'agente. Questo processo utilizza il parametro `AgentFOVDegrees` proveniente dal modulo **Agent** (Capitolo 5.2.3).

Se, per qualche motivo, fosse necessario ottenere una lista di cartelli visibili, ma non necessariamente nel campo visivo dell'agente, la classe tiene aggiornata una lista separata, denominata **SignboardsAround**.

### 5.2.6 Simulation Agent

Il modulo **Simulation Agent** permette di raccogliere dati sulla segnaletica dagli agenti durante il loro percorso all'interno dell'edificio.

Esso sfrutta la componente *Signboard Aware Agent* per monitorare l'ingresso e l'uscita dell'agente dalle aree di visibilità dei cartelli.

Quando l'agente entra in una *VCA*, viene registrato l'istante di ingresso. Al momento dell'uscita, viene calcolato il **tempo di permanenza** all'interno dell'area, sottraendo l'ora di entrata a quella di uscita. Se il tempo di permanenza è superiore al parametro del cartello **MinimumReadingTime**, descritto nel Capitolo 2.2, l'evento viene registrato come una visita al cartello.

Monitorando il numero di visite registrate per ogni cartello, sarà successivamente possibile valutare l'efficacia del posizionamento della segnaletica, fornendo informazioni utili per migliorare la disposizione e l'accessibilità della segnaletica all'interno dell'edificio.

### 5.2.7 Wanderer Agent

**Wanderer Agent** è il componente principale che permette ad un agente di muoversi liberamente all'interno di un edificio, alla ricerca di un traguardo preimpostato, usando la segnaletica a disposizione nell'edificio.

Esso permette di effettuare valutazioni sulla posizione segnaletica, confrontando il tempo impiegato e la lunghezza del tragitto che l'agente percorre.

Per operare correttamente, il modulo necessita di una serie di **goal** (traguardi), che vengono memorizzati in una coda. L'agente si muove poi liberamente all'interno dell'edificio, facendo attenzione alle informazioni fornite dalla segnaletica, per raggiungere i traguardi predefiniti. Al raggiungimento di ogni traguardo, l'agente procede immediatamente verso il successivo. Una volta completato l'ultimo goal, l'agente viene rimosso dall'ambiente.

La classe mette a disposizione il seguente evento:

```
1 public delegate void OnGoalReached(bool isLastGoal, bool
  success);
2 public event OnGoalReached GoalReachedEvent;
```

Esso viene invocato ogni volta che l'agente raggiunge il traguardo corrente. L'evento specifica se il traguardo è l'ultimo della lista oppure se la ricerca è stata abbandonata risultando in un fallimento.

Questo componente implementa l'interfaccia **IAgentWithGoal** definita come segue:

```
1 public interface IAgentWithGoal {
2     public void AddGoal(IRouteMarker goal);
3     public IRouteMarker CurrentGoal();
4     public int GoalCount();
5     public IRouteMarker RemoveCurrentGoal();
6     public void ClearGoals();
7     public void StartTasks();
8 }
```

I metodi dell'interfaccia sono autoesplicativi: permettono di aggiungere, rimuovere e gestire i traguardi nella coda, nonché di avviare il percorso dell'agente attraverso il metodo

`StartTasks()`.

Attualmente, essa supporta solo **IRouteMarker** come *goal*, che include sia gli *IntermediateMarker* che i *LinkedMarker* introdotti nel Capitolo 4.6, ma questo potrebbe cambiare in futuro.

**IAgentWithGoal** viene fornita in quanto copre un ruolo fondamentale nell'architettura, ovvero la possibilità di assegnare diversi traguardi ad un agente.

Sebbene, al momento della stesura di questo documento, non viene usata da un elevato numero di componenti, essa faciliterà notevolmente l'espansione e l'uso dell'architettura dei vari agenti in futuro.

## Logica di movimento

Il movimento di tipo “*vagante*” dell’agente è regolato da una **macchina a stati finiti** (o *FSM*<sup>2</sup>) che gli consente di prendere decisioni basate sulla configurazione dell’ambiente circostante.

Questo approccio trae ispirazione dal lavoro di Dubey et al. [5]. L’articolo in questione impiega un modello basato sulla *teoria dell’informazione* per simulare l’orientamento degli agenti, che dista notevolmente dal metodo utilizzato in questo lavoro.

Tuttavia, la struttura della macchina a stati finiti da essi proposta si adatta sorprendentemente bene agli agenti di questa sezione, ma è importante sottolineare che, sebbene la struttura sia ispirata da quella originale, l’implementazione degli stati è completamente originale.

---

<sup>2</sup>Finite State Machine

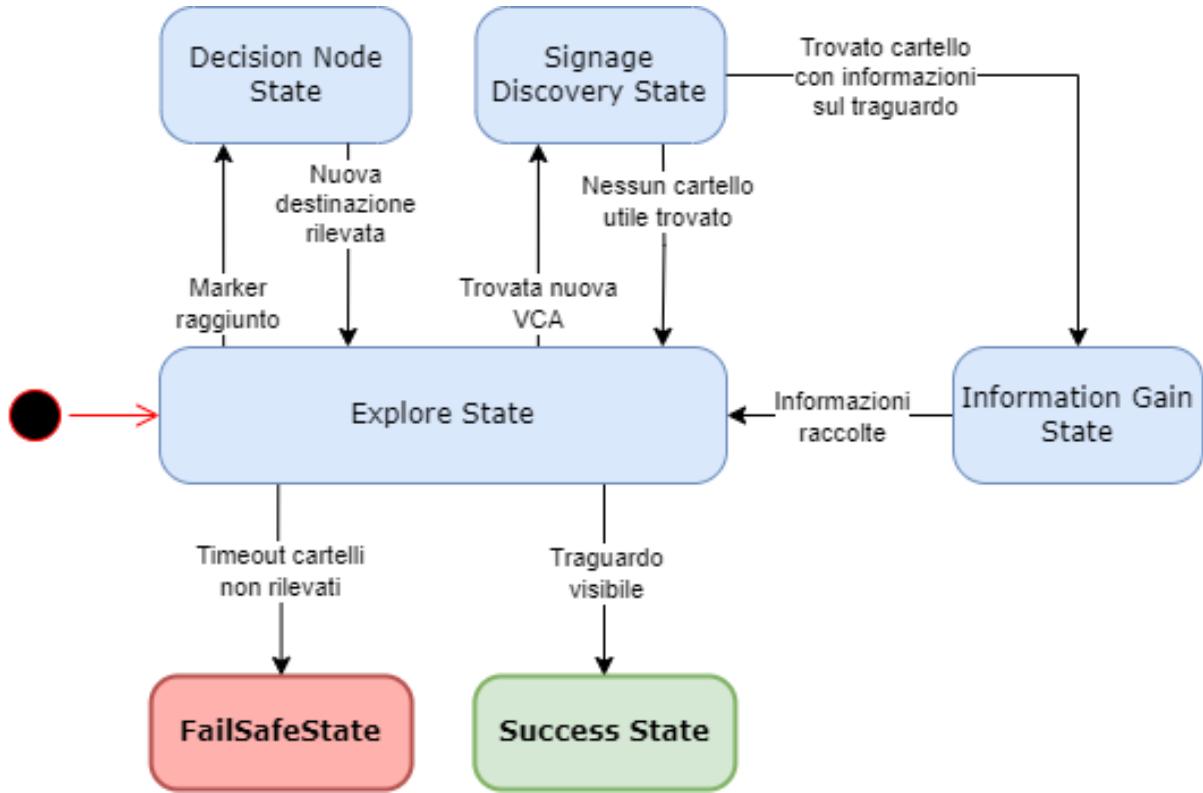


Figura 5.3: Macchina a stati finiti di Wanderer Agent.

Di seguito verranno esplorati i singoli stati nel dettaglio.

**Explore State** *Agent Wanderer* esplora l'ambiente muovendosi tra i vari marker. Quando entra nello stato di esplorazione, l'agente procede verso la prossima destinazione intermedia, che inizialmente è il marker più vicino. Le destinazioni successive sono determinate dallo stato *Decision Node*.

Durante il movimento l'agente è informato sulla segnaletica circostante e potrebbe decidere di interrompere temporaneamente il percorso per avvicinarsi e acquisire ulteriori informazioni.

**Decision Node State** Quando l'agente raggiunge una destinazione intermedia, entra nello stato di decisione, in cui valuta la prossima destinazione. In questo momento viene decisa la destinazione intermedia successiva, valutando tutti i marker immediatamente raggiungibili dalla posizione dell'agente.

Ad ognuno di essi viene assegnato un peso, seguendo una lista di regole:

- Se ha appreso delle indicazioni dalla segnaletica, l'agente preferisce seguire la direzione indicata

- L’agente preferisce evitare di tornare indietro
- L’agente preferisce marker più vicini
- L’agente preferisce raggiungere marker che ha visitato meno volte
- L’agente preferisce evitare di attraversare altri marker per raggiungere la propria destinazione

Queste regole non hanno uguale influenza sulla decisione ed essa può essere modificata attraverso dei parametri.

Una volta assegnato un peso ad ogni destinazione considerata, questa viene scelta casualmente con una probabilità direttamente proporzionale al peso.

La destinazione viene salvata e l’agente torna nello stato di esplorazione per cercare di raggiungerla.

**Signage Discovery State** Se durante l’esplorazione l’agente entra nella area di visibilità di un cartello che non ha ancora visitato, passa allo stato **Signage Discovery**. Qui si avvicina al cartello per acquisire informazioni. Se queste sono utili per il traguardo corrente, passa allo stato **Information Gain**; altrimenti, ritorna allo stato **Decision Node** per continuare l’esplorazione.

**Information Gain State** In questo stato, l’agente cerca di ottenere indicazioni utili sulla posizione del traguardo.

Per questo scopo è stato sviluppata la classe **SignboardDirections**, che è possibile applicare ad un qualsiasi elemento di segnaletica per specificare che questo contiene indicazioni. Ogni istanza di **SignboardDirections** è possibile specificare i traguardi a cui le indicazioni fanno riferimento.

Se l’agente incontra un cartello con indicazioni rilevanti per il traguardo corrente, riceve la posizione del primo marker sul percorso più breve per raggiungere tale obiettivo. Nel caso in cui non ci siano marker intermedi, l’agente riceve direttamente la posizione del traguardo.

Oltre alle indicazioni sulla posizione, l’agente riceve anche una “**direzione preferita**”, ovvero un vettore direzione che punta verso la posizione del marker appena ricevuto.

Questa direzione preferita ha un’influenza significativa sulle decisioni prese nei successivi *Decision Node State*, aiutando l’agente a orientarsi meglio. Tuttavia, per evitare che l’agente rimanga eccessivamente focalizzato su un’unica area, la direzione preferita perde la sua influenza nel tempo, in quanto non è garantito che questa punti direttamente al

traguardo.

L’agente perde immediatamente la direzione preferita quando attraversa un punto significativo dell’ambiente, come l’entrata di una scalinata, che lo porta a cambiare piano.

**Fail Safe State** Se l’agente rimane per troppo tempo nello stato di esplorazione senza trovare il traguardo, entra nello stato **Fail Safe**. Ciò significa che, dopo una lunga ricerca infruttuosa, l’agente abbandona l’obiettivo corrente e passa a cercare il prossimo nella lista.

**Success State** Se durante la fase di esplorazione viene individuato il traguardo, l’agente lo raggiunge, completando il proprio compito. Se ci sono ulteriori destinazioni da raggiungere, l’agente procede; in caso contrario, viene rimosso dall’ambiente.

### 5.2.8 Routed Agent

**Routed Agent** è una versione semplificata di *Wanderer Agent*.

Esso si limita a ricevere una serie di traguardi da raggiungere, esattamente come *Wanderer Agent*, ma esso cerca di raggiungerli senza alcuna logica complessa, seguendo il percorso più breve tra essi.

Anch’esso implementa l’interfaccia **IAgentWithGoal** per una maggiore flessibilità, nel caso si voglia introdurre altri moduli simili in futuro.

## 5.3 Introduzione degli agenti nell’ambiente

**Agent Handler** è il modulo che si occupa di gestire la presenza degli agenti nella scena. Esso tiene traccia di tutti gli agenti presenti, fornendo una lista aggiornata a cui è possibile accedere in qualsiasi momento.

Gli agenti vengono introdotti nella scena attraverso le cosiddette **Aree di spawn** (Figura 5.4), aree ridimensionabili che consentono di generare agenti all’interno dell’edificio, posizionandoli sopra l’area definita.

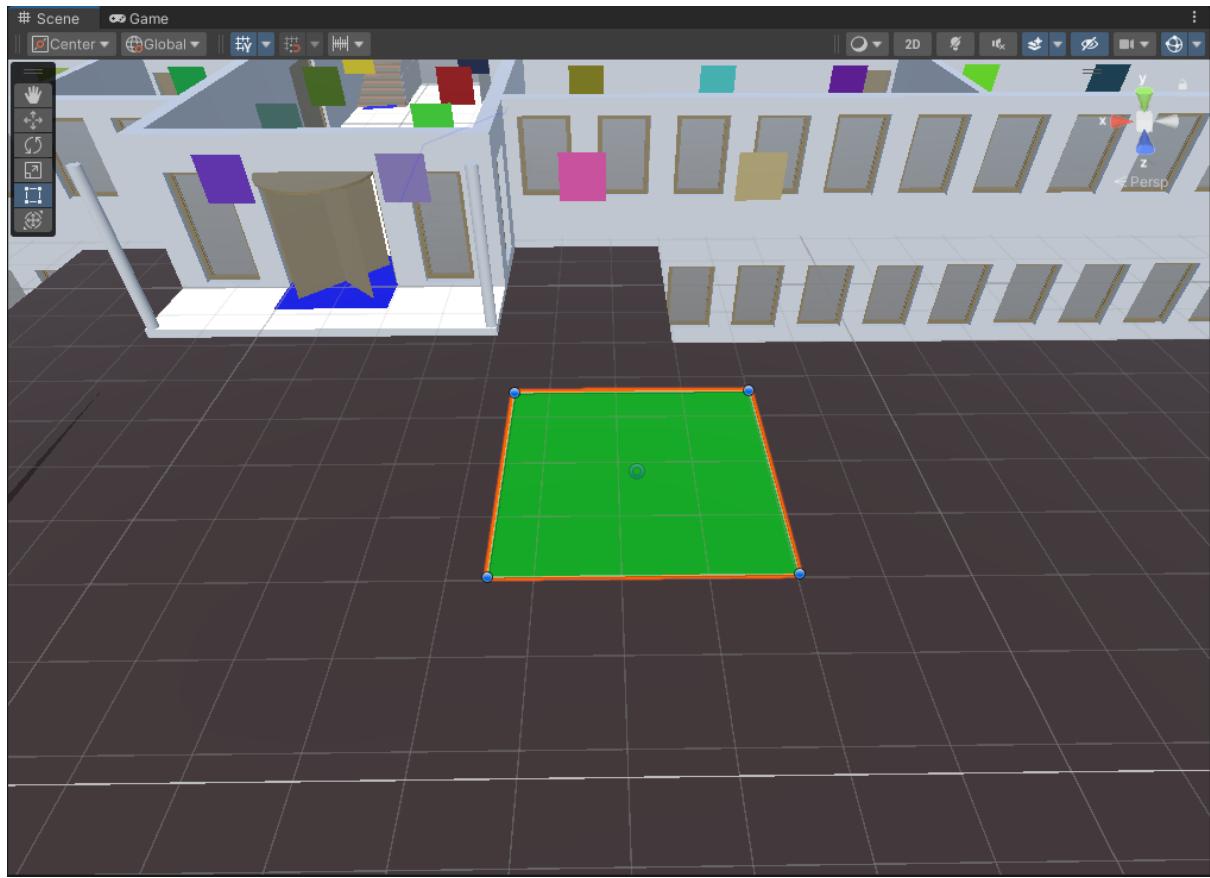


Figura 5.4: Esempio di Spawn Area (in verde) posizionata all'ingresso di un edificio.

Esistono diverse implementazioni di queste aree, ma fanno tutte riferimento alla classe astratta **SpawnAreaBase**, che offre funzionalità dedicate all'introduzione di agenti nell'ambiente.

In particolare è possibile:

1. Avviare o fermare lo spawn di agenti.
2. Modificare la frequenza di spawn.
3. Scegliere se posizionare gli agenti in modo casuale all'interno dell'area.
4. Limitare il numero di agenti da generare.

La gestione della posizione di spawn è cruciale quando si imposta un'alta frequenza di spawn o si genera un numero elevato di agenti. In questi casi, c'è il rischio che un agente venga creato troppo vicino a un altro, causando un comportamento indesiderato nel motore fisico di Unity (ad esempio, la repulsione violenta tra gli agenti). Per evitare questo problema, si imposta una distanza minima tra gli agenti già presenti e quelli in fase di generazione. Se non si trova una posizione libera dopo diversi tentativi, lo spawn viene

annullato fino al tick successivo.

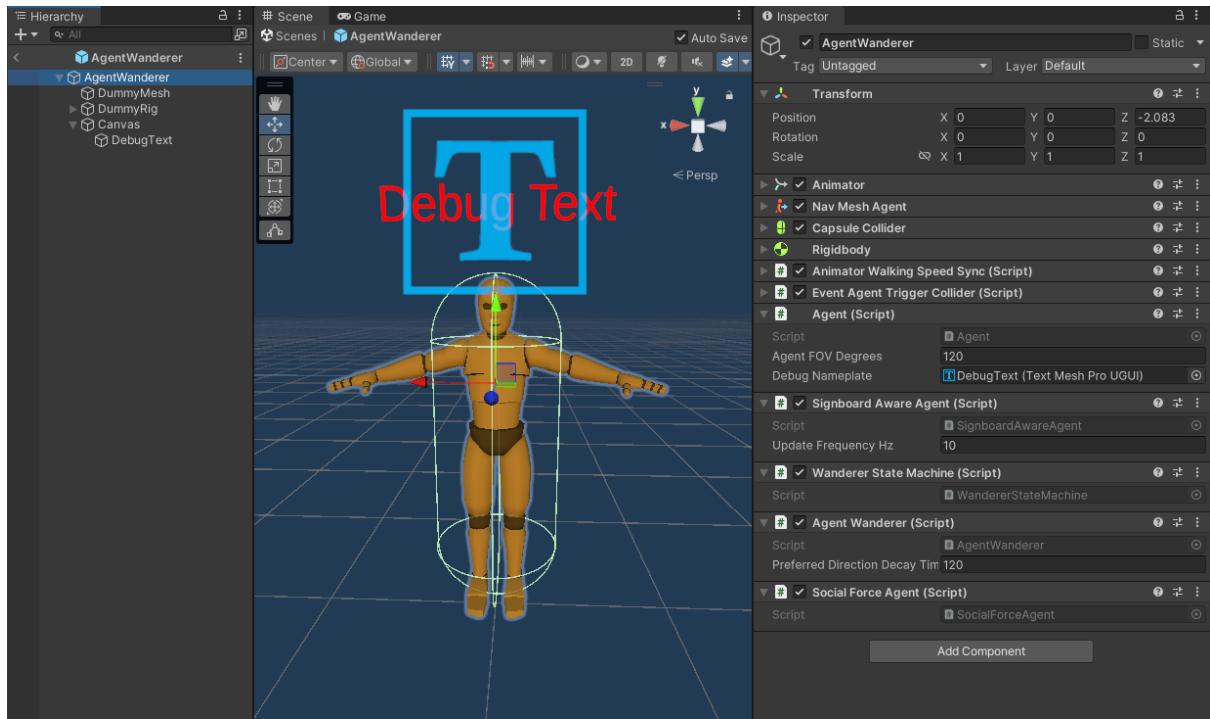


Figura 5.5: Esempio di prefabbricato di un agente (*Agent Wanderer*). A destra è visibile l'elenco dei moduli ad esso collegati.

Lo spawn del singolo agente viene gestito attraverso il sistema di **prefabbricati** ( comunemente chiamati “*Prefab*”) di Unity. Esso permette di salvare come file un’istanza di un qualsiasi *GameObject*<sup>3</sup> ed usarlo come base per creare nuove copie a partire dall’oggetto salvato (Figura 5.5). Ogni nuova istanza del prefabbricato avrà gli stessi moduli e parametri ad essi associati.

Nella sua interfaccia (Figura 5.6), ogni area mette a disposizione il parametro che permette di specificare quale prefabbricato usare, ma alcune implementazioni potrebbero richiedere che questo includa componenti specifici.

<sup>3</sup>Classe alla base di qualsiasi oggetto in una scena di Unity.  
<https://docs.unity3d.com/ScriptReference/GameObject.html>

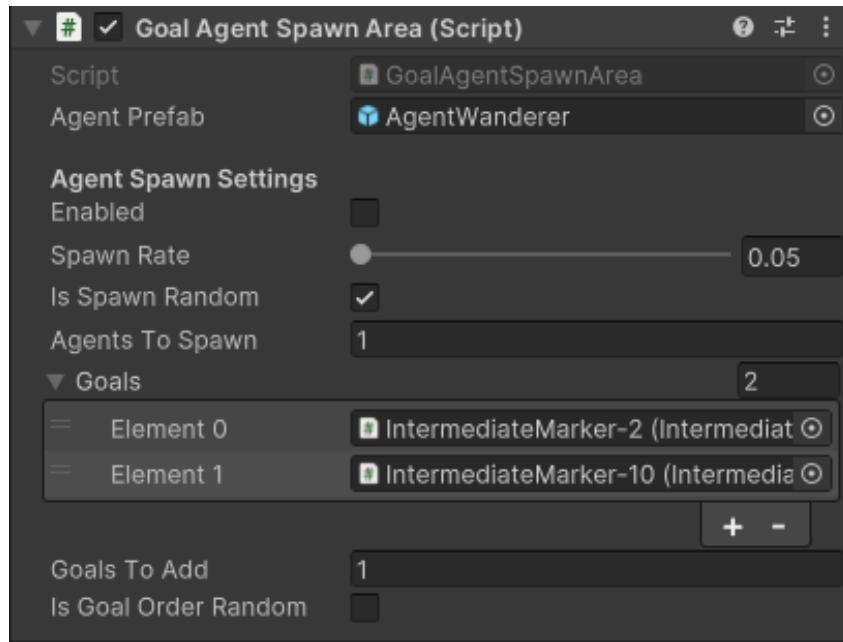


Figura 5.6: Interfaccia di GoalAgentSpawnArea.

Le implementazioni di **SpawnAreaBase** disponibili al momento sono:

- **SpawnArea**: Consente di specificare una lista di altri oggetti di tipo `SpawnAreaBase`, che verranno scelti casualmente come destinazione per gli agenti istanziati da questa area.
- **TestSpawnArea**: Genera un nuovo agente ogni volta che si fa clic su un punto dell’edificio, utilizzandolo come destinazione. Utile per il debug, insieme a `SpawnArea`.
- **GoalAgentSpawnArea**: Permette di generare agenti che implementano l’interfaccia `IAgentWithGoal`. Consente di specificare quali traguardi saranno assegnati agli agenti e se l’ordine deve essere mantenuto o randomizzato.
- **CPTRoutedSpawnArea**: Richiede che il prefabbricato da generare abbia associato il modulo `RoutedAgent` (Capitolo 5.2.8). Esso genera agenti che seguono la coda di marker proveniente dal modulo trattato nel Capitolo 7.2.

# Capitolo 6

## Validazioni

In questo capitolo, verranno analizzati alcuni aspetti del progetto, focalizzandosi sulle funzionalità sviluppate e sulla loro conformità rispetto agli studi esistenti. È importante precisare che non è stato creato un modello di simulazione predefinito, ma piuttosto uno strumento flessibile che consente all'utente di definire i propri modelli.

Grazie alla vasta gamma di parametri configurabili, i risultati delle simulazioni dipendono fortemente dalle competenze e dall'esperienza di chi utilizza lo strumento.

Per sviluppare un metodo oggettivo che valuti completamente lo strumento in tutte le sue sfaccettature, sarebbero necessarie risorse significative. Nello specifico, occorrerebbe condurre diversi test empirici in edifici reali, per i quali siano disponibili modelli BIM accurati e completi. Tali risorse possono essere difficili da reperire, poiché implicano un considerevole investimento di tempo e infrastruttura.

Ciò che possiamo fare è concentrarci sulla validazione di componenti specifiche dello strumento, derivate da studi precedenti. Per queste funzionalità, è possibile confrontare i risultati con quelli riportati dagli sviluppatori originali, laddove disponibili. Questo tipo di confronto ci consente di verificare che le implementazioni siano coerenti con gli standard esistenti e producano risultati affidabili rispetto a quelli presenti in letteratura.

### 6.1 Validazione del Social Force Agent

Il modello proposto da Mehdi Moussaïd et al. [16] per la simulazione delle forze sociali (Capitolo 3.2.2) si basa su test empirici. I ricercatori hanno osservato il comportamento di pedoni in un corridoio, tracciando le loro scelte e traiettorie.

I test si sono svolti in un corridoio lungo 7.88 metri e largo 1.75 metri, in tre diverse condizioni:

1. **Condizione 1:** A un singolo pedone è stato chiesto di camminare avanti e indietro nel corridoio per 3 minuti.
2. **Condizione 2:** Un soggetto rimaneva fermo al centro del corridoio, mentre un altro seguiva le stesse istruzioni della Condizione 1, evitando l'ostacolo. Ogni partecipante ha eseguito una volta il test in movimento e una volta come “ostacolo”, con ogni replica della durata di 3 minuti.
3. **Condizione 3:** Due soggetti hanno ricevuto le stesse istruzioni della condizione uno, ma partendo da estremità opposte del corridoio, dovendo eludersi a vicenda. Per ogni nuova prova veniva dato un segnale di partenza, in modo che i pedoni si incontrassero sempre al centro del corridoio.

### 6.1.1 Confronto tra i risultati

I test in **Condizione 1** hanno permesso di determinare i parametri relativi all’accelerazione e alla velocità massima dei pedoni, implementati facilmente nell’ambiente Unity grazie agli strumenti forniti dal motore stesso. Pertanto, non è necessaria una validazione specifica per questa condizione.

Per effettuare i test di validazione nelle **Condizioni 2 e 3**, è stato ricreato in 3D l’ambiente di test mantenendo le dimensioni originali ed è stato importato in Unity. Figura 6.1).

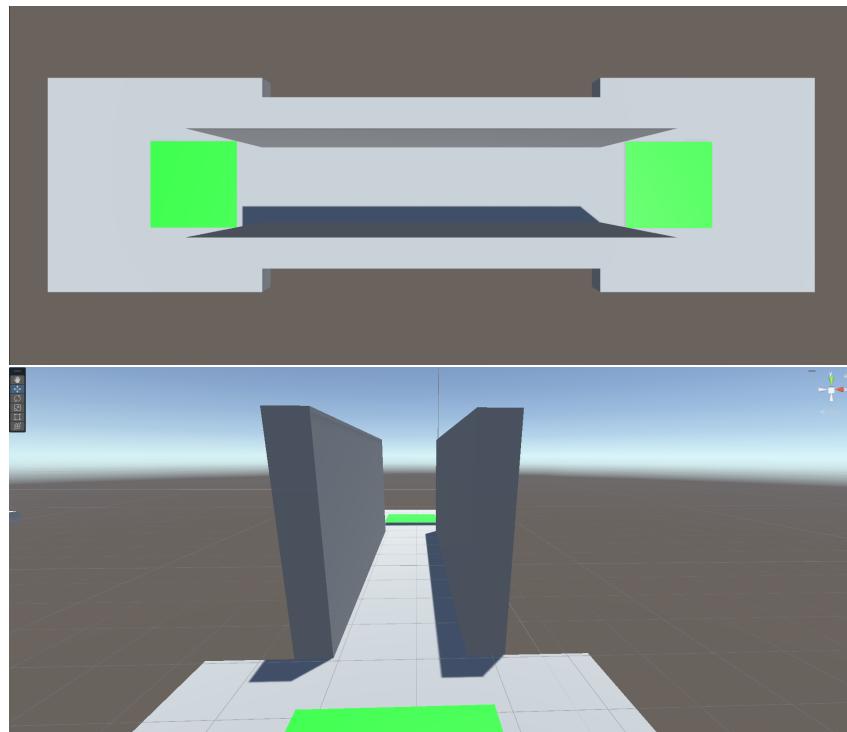


Figura 6.1: Rappresentazione tridimensionale del corridoio usato per i test.

Possiamo confrontare i risultati ottenuti da Mehdi Moussaïd et al. [16] (Figura 6.2) con quelli della nostra implementazione (Figura 6.3). In entrambe le rappresentazioni, sulla sinistra viene mostrata la media delle traiettorie su 1000 simulazioni e la scelta di evitare l'ostacolo andando a sinistra o a destra.

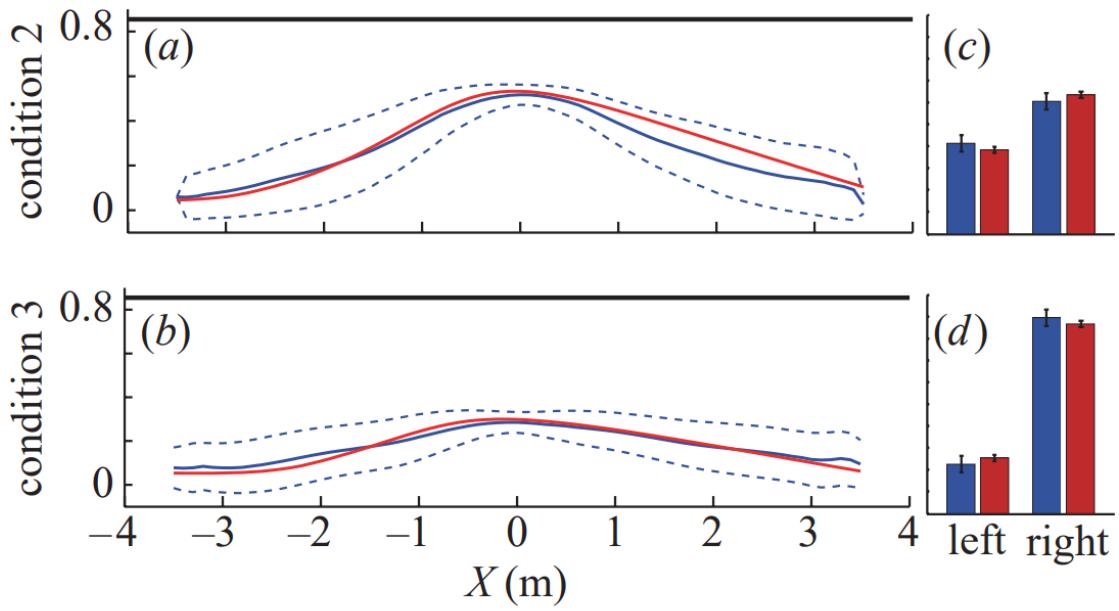


Figura 6.2: Risultati originali di Mehdi Moussaïd et al. [16]. In alto: Test in condizione 2. In basso: Test in condizione 3.

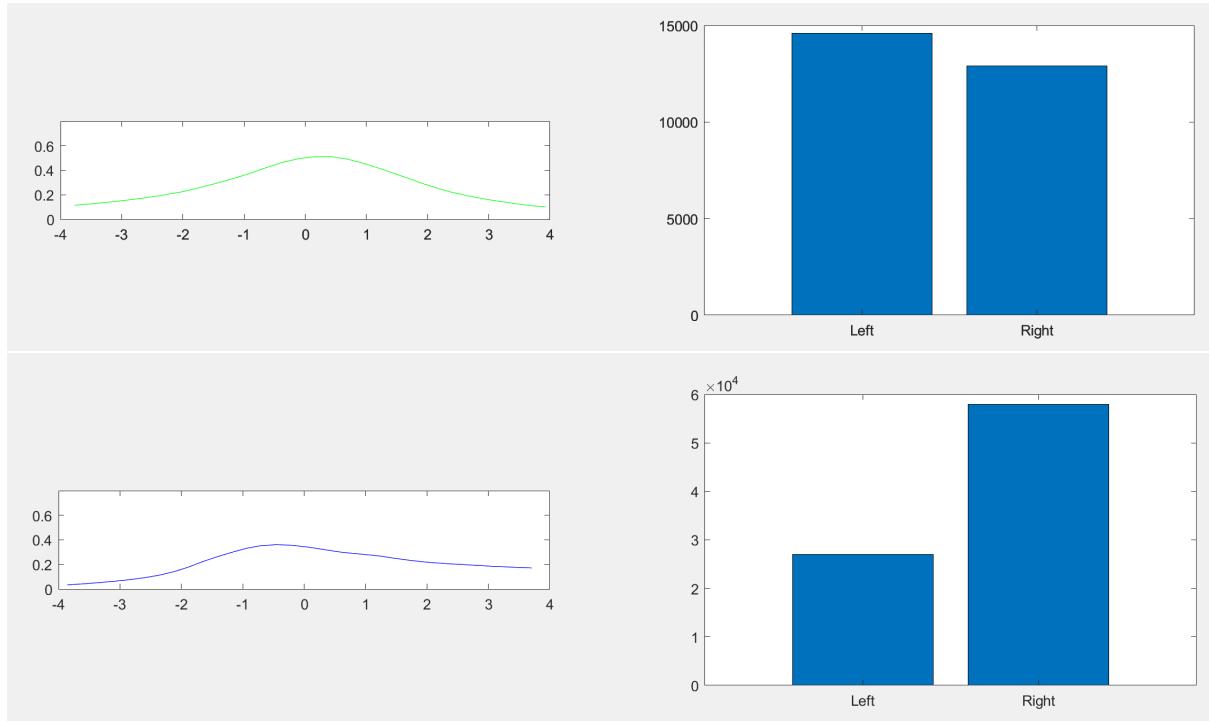


Figura 6.3: Risultati della nostra implementazione. In alto: Test in condizione 2. In basso: Test in condizione 3.

Come si può osservare, le traiettorie generate dalla nostra simulazione mostrano una notevole somiglianza con quelle del modello originale.

In **Condizione 3** entrambe le simulazioni evidenziano un incremento significativo nel

numero di pedoni che scelgono di evitare l'ostacolo andando a destra, a causa dell'introduzione intenzionale di un **bias** nel modello.

Per quanto riguarda la **Condizione 2**, l'effetto del bias è praticamente inesistente. In questo caso, la decisione di aggirare l'ostacolo statico passando a destra o a sinistra dipende principalmente dalla posizione di partenza del pedone, che non è sempre centrale ma varia casualmente lungo l'inizio del corridoio.

## 6.2 Validazione Wanderer Agent

Per verificare l'efficacia del modulo **Wanderer Agent** (Capitolo 5.2.7), sono stati sviluppati due agenti di confronto: **Perfect Agent** e **SignBlind Agent**. L'obiettivo è di comprendere dove il Wanderer Agent si collochi in termini di performance, analizzando come si comporta rispetto agli agenti che rappresentano gli estremi dello spettro di navigazione.

### 6.2.1 Agenti di confronto

Due agenti derivati dal Wanderer Agent sono stati implementati per la validazione e il confronto. Questi agenti non solo permettono di valutare le prestazioni del Wanderer Agent, ma possono essere utilizzati anche per testare altri tipi di agenti che potrebbero essere introdotti in futuro.

#### Perfect Agent

**Perfect Agent** è progettato per rappresentare un individuo con conoscenze pregresse dell'ambiente. Questo agente raggiunge i traguardi direttamente, seguendo il percorso più breve possibile. Il suo comportamento simula quello di un pedone che conosce già la mappa dell'edificio e non necessita di segnaletica per orientarsi.

Come tutti gli agenti discussi in questa sezione, esso riceve una lista di traguardi da raggiungere ed usa i percorsi generati dal modulo **NavMesh** (Capitolo 4.3) per raggiungerli nel minor tempo possibile.

#### SignBlind Agent

**SignBlind Agent** si comporta in modo molto simile al Wanderer Agent, ma con la differenza cruciale che ignora completamente la segnaletica presente nell'edificio. Questo

agente è utile per valutare come la presenza della segnaletica influisca sulle prestazioni rispetto a un pedone che si muove senza utilizzare informazioni visive fornite dai cartelli.

La sua implementazione utilizza una macchina a stati finiti simile a quella del Wanderer Agent, ma priva degli stati legati all’interazione con la segnaletica (Figura 6.4). Sebbene sia ancora in grado di prendere decisioni locali tramite il **Decision Node State**, queste non saranno mai influenzate dalle informazioni presenti nei cartelli.

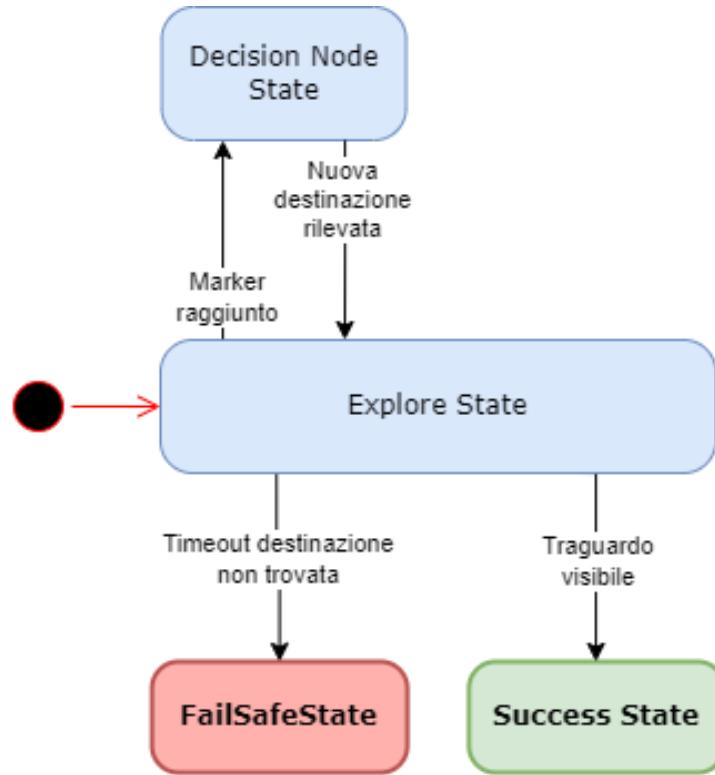


Figura 6.4: Macchina a stati finiti di SignBlind Agent.

### 6.2.2 Confronto delle performance degli agenti

In questa sezione, mettiamo a confronto le performance del Wanderer Agent con quelle del Perfect Agent e del SignBlind Agent. Il confronto viene effettuato in termini di lunghezza del percorso, il tempo impiegato e la percentuale di successo nel raggiungere tutti gli obiettivi, considerando diversi scenari di segnaletica.

Ogni test è stato eseguito inviando 20 agenti dello stesso tipo (Wanderer, Perfect o SignBlind), ciascuno con tre traguardi da raggiungere. I traguardi sono stati selezionati casualmente tra i marker presenti nell’edificio, ma a tutti gli agenti è stata assegnata la stessa sequenza di obiettivi, nello stesso ordine.

I dati presentati, relativi al tempo e spazio impiegato, corrispondono alla media dei risultati ottenuti dagli agenti che sono riusciti a raggiungere *tutti* i traguardi.

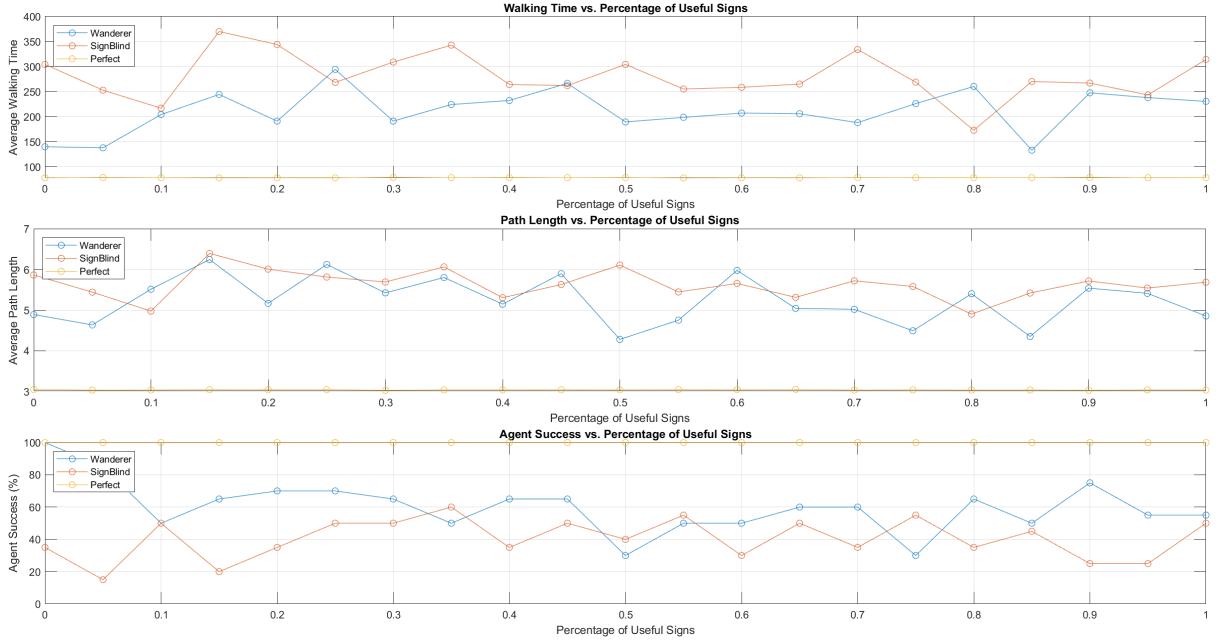


Figura 6.5: Confronto delle performance tra i tre tipi di agente. I alto il test che mette a confronto il tempo impiegato, al centro la lunghezza del percorso ed in basso la percentuale di successi.

**Perfect Agent** ha mostrato il comportamento prevedibile: non ha avuto necessità di fermarsi per raccogliere informazioni sui traguardi, e ogni agente ha completato con successo tutti gli obiettivi nel minor tempo possibile, seguendo sempre il percorso più breve.

Il **Wanderer Agent**, invece, ha presentato una performance media molto simile a quella del **SignBlind Agent** per quanto riguarda la lunghezza del percorso. Tuttavia, si osserva un miglioramento significativo nei tempi di percorrenza e nella percentuale di successo nel raggiungere tutti i traguardi, evidenziando un vantaggio concreto nell'uso della segnaletica. Questo vantaggio, però, è evidente in modo netto in quanto il test non ha coinvolto l'intero edificio, il che ha dato a SignBlind Agent maggiori probabilità di imbattersi casualmente nei traguardi.

Un altro aspetto interessante è che, quando la quantità di segnaletica diventa eccessiva (superiore al 70%), il Wanderer Agent perde parte del proprio vantaggio. L'agente, infatti, si trova costretto a fermarsi frequentemente per seguire le indicazioni della maggior parte dei cartelli, il che incrementa significativamente il tempo complessivo del percorso.

Anche se non osserviamo un miglioramento netto e consistente tra l'agente che ignora

completamente la segnaletica e quello che la segue, è evidente che la presenza della segnaletica ha un impatto positivo, riducendo il tempo di percorrenza e aumentando la percentuale di successi nel raggiungimento dei traguardi.

# Capitolo 7

## Valutazione della segnaletica

In questo capitolo verranno proposti degli esempi di utilizzo dello strumento sviluppato, facendo uso degli elementi introdotti precedentemente, per effettuare valutazione qualitative sulla segnaletica. Questi esempi dimostreranno come l'integrazione dei moduli di simulazione e navigazione possa essere utilizzata per ottimizzare il posizionamento della segnaletica.

I test che andranno eseguire sono stati eseguiti all'interno del **Karlsruhe Institute of Technology (KIT)** (Figura 7.1, che mette liberamente a disposizione il modello IFC di uno dei loro edifici [10]).



Figura 7.1: Come si presenta il modello 3D del *Karlsruhe Institute of Technology* all'interno dell'editor di Unity.

Le simulazioni sono state limitate ai primi due piani dell'edificio, a causa delle limitazioni hardware del calcolatore utilizzato, che presenta le seguenti specifiche tecniche:

- **CPU:** AMD Ryzen 5 3600
- **RAM:** 32 GB
- **GPU:** Nvidia Geforce GTX 1650

Questa configurazione, sebbene adeguata per la maggior parte delle simulazioni di base, limita il numero di agenti e la complessità del modello che può essere gestito simultaneamente. Tuttavia, la scelta di limitare le dimensioni dell’ambiente ha permesso di mantenere un livello di dettaglio sufficiente per effettuare analisi significative sulle interazioni tra agenti e segnaletica.

**Parametri della segnaletica** Per tutte le simulazioni, i parametri della segnaletica sono stati impostati come segue:

- Viewing Distance: 30
- Viewing Angle: 60
- Minimum Reading Time: 1s

**Tipi di agenti** Sono stati introdotti due tipi di agenti, che differiscono per l’altezza degli occhi:

- Adulto - Maschio: 1.75m
- Persona in sedia a rotelle: 1.25m

Questi parametri sono stati scelti in modo arbitrario, prendendo ispirazione dai test effettuati da Motamed et al. [15]. Questi valori non sono particolarmente importanti per il momento, in quanto andremo a confrontare cartelli con parametri identici. Il normale impiego sul campo richiederà test empirici con la segnaletica che si intende usare e gli individui che si intende ospitare.

## 7.1 Analisi Statica

Dopo aver introdotto la segnaletica dell’edificio, è possibile ricavare un dato preliminare, denominato **Coverage**. La **Coverage** è un punteggio che varia tra 0 e 1, che fa riferimento ad uno specifico tipo di agente, direttamente proporzionale all’area di *Visibility Plane* coperta dall’area di visibilità del cartello. Maggiore è il valore di coverage, migliore è la copertura visiva del cartello per quel tipo di agente nell’ambiente.

Le informazioni relative al coverage possono essere esportate in formato CSV<sup>1</sup> (Tabella 7.1 e Tabella 7.2). Da questo file è poi generare grafici come quello in Figura 7.2, che mostra quali sono gli elementi di segnaletica con punteggio di coverage più alto, tenendo conto di tutti gli agenti.

Il nome di ciascun cartello include la sua posizione all'interno della griglia generata dal modulo Signboard Grid Generator (Capitolo 4.5), mentre tra parentesi è indicato il piano dell'edificio in cui si trova il cartello, poiché ogni piano ha la sua griglia associata.

Tabella 7.1: Coverage per *Adult - Male*

Nome	Coverage[Adult - Male]
Signboard [6, 5](Decke-001)	1
Signboard [6, 4](Decke-001)	0,7741658
Signboard [7, 4](Decke-001)	0,7708978
Signboard [11, 2](Decke-002)	0,7105263
Signboard [12, 2](Decke-002)	0,7079464
Signboard [10, 2](Decke-002)	0,5306157
Signboard [13, 2](Decke-002)	0,5079119
Signboard [6, 3](Decke-001)	0,501376
Signboard [7, 3](Decke-001)	0,498108
Signboard [6, 3](Decke-002)	0,3837289
Signboard [12, 1](Decke-002)	0,246474
Signboard [11, 1](Decke-002)	0,246474
Signboard [6, 2](Decke-001)	0,2420021
Signboard [7, 2](Decke-001)	0,2411421
Signboard [10, 1](Decke-002)	0,2119023

---

<sup>1</sup>Un CSV (comma-separated values) è un file con valori separati da virgole che consente di salvare i dati in formato tabellare.

Tabella 7.2: Coverage per *Wheelchair User*

Nome	Coverage[Wheelchair User]
Signboard [6, 5](Decke-001)	1
Signboard [6, 4](Decke-001)	0,7854015
Signboard [7, 4](Decke-001)	0,7826642
Signboard [11, 2](Decke-002)	0,7085766
Signboard [12, 2](Decke-002)	0,7054744
Signboard [10, 2](Decke-002)	0,525
Signboard [13, 2](Decke-002)	0,5029197
Signboard [6, 3](Decke-001)	0,4972628
Signboard [7, 3](Decke-001)	0,4937956
Signboard [6, 3](Decke-002)	0,3691606
Signboard [6, 2](Decke-001)	0,2240876
Signboard [7, 2](Decke-001)	0,2237226
Signboard [11, 1](Decke-002)	0,2235401
Signboard [12, 1](Decke-002)	0,2235401
Signboard [10, 1](Decke-002)	0,1916058

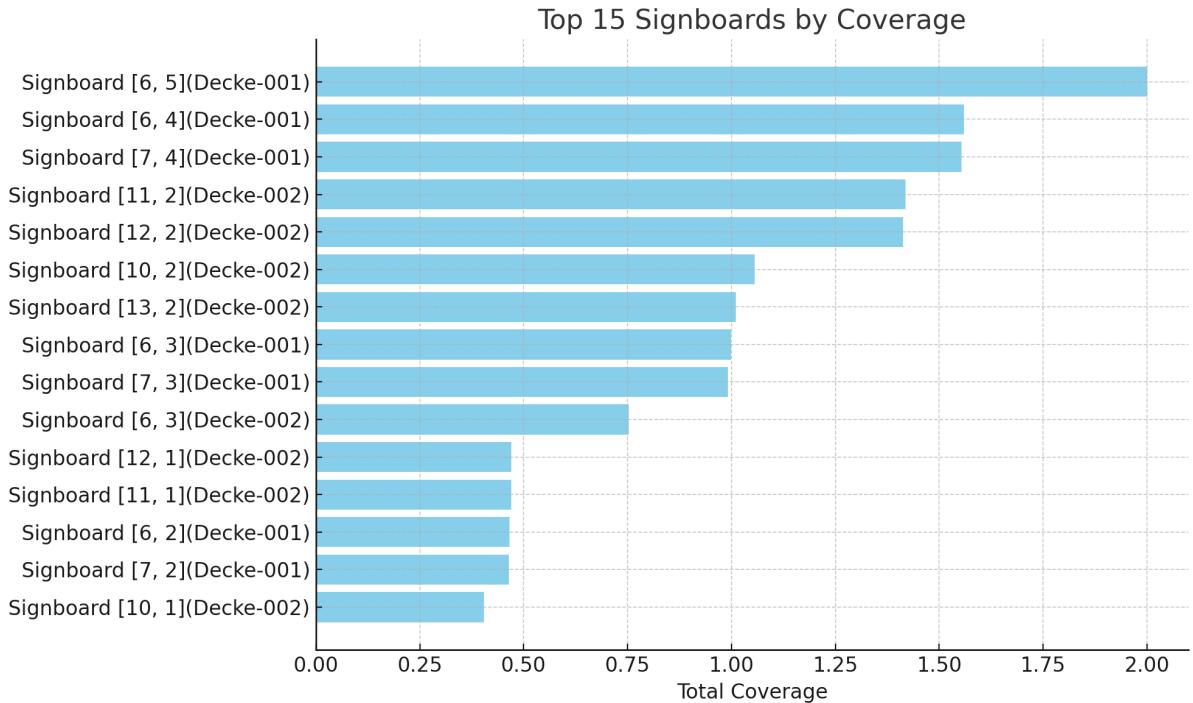


Figura 7.2: Barplot della coverage totale delle prime 15 *Signboard* con coverage più alta.

Queste informazioni preliminari possono essere utilizzate per prendere decisioni rapide

sulla disposizione della segnaletica, senza necessariamente dover tenere conto del flusso pedonale. In particolare, la coverage può essere utilizzata per selezionare in anticipo i cartelli con punteggi più bassi, che potrebbero richiedere un miglior posizionamento o essere rimossi completamente per migliorare le performance della simulazione.

## 7.2 Ricerca dell'ottimo

In questa sezione viene presentato un metodo per identificare le posizioni ottimali per la segnaletica, tenendo conto del flusso pedonale all'interno dell'edificio.

È possibile formulare l'ipotesi che, in un edificio trafficato, la segnaletica con maggior grado di visibilità è quella che gli utenti notano più spesso durante il loro percorso. Per questo motivo possiamo fare uso del modulo **Simulation Agent** (Capitolo 5.2.6), che tiene traccia dei cartelli incontrati durante il suo periodo di attività. Questo modulo ci permette di raccogliere dati su quali cartelli vengono notati più frequentemente, offrendo un'indicazione su quali siano le posizioni più visibili e quindi più efficaci per la segnaletica.

### 7.2.1 Rete di marker e percorso ottimale

Uno dei problemi principali nella simulazione è garantire che i pedoni simulati coprano uniformemente l'intero edificio, in modo che ogni cartello abbia la stessa probabilità di essere visto. Per fare ciò, possiamo sfruttare la rete di marker introdotta nel Capitolo 4.6, che può essere immaginata come un grafo che si estende per tutto l'edificio. Nella Figura 7.3 viene mostrata una rappresentazione visiva di questo concetto, dove i nodi rappresentano i marker e gli archi collegano i marker direttamente raggiungibili.



Figura 7.3: Rappresentazione grafica degli archi del grafo che collegano i vari marker. I muri esterni sono stati nascosti per migliorare la visibilità dei nodi.

Per garantire che gli agenti percorrano tutti i percorsi possibili tra i marker e, di conseguenza, abbiano la possibilità di vedere tutti i cartelli, possiamo applicare la soluzione al problema del **Postino Cinese orientato** o **CPT**<sup>2</sup> [19]. Questo problema matematico permette di trovare un percorso che tocca tutti gli archi di un grafo, consentendo il passaggio ripetuto sugli stessi archi.

Utilizzando questa soluzione, siamo in grado di generare un percorso che copre tutte le aree dell’edificio, garantendo un’alta probabilità che ogni cartello venga visualizzato almeno una volta dagli agenti, a meno che non sia completamente invisibile ad una certa categoria di agenti.

Permettendo agli agenti di tornare sui propri passi, promuoviamo inoltre l’interazione tra di essi, aumentando le possibilità di incroci e sovrapposizioni tra i loro percorsi, il che offre una simulazione più realistica del flusso pedonale.

Per introdurre gli agenti nell’ambiente utilizzeremo **CPTRoutedSpawnArea**, introdotta nel capitolo 5.3. Questa particolare area di spawn consente di assegnare agli agenti che utilizzano il componente **RoutedAgent** un percorso generato dal modulo **RoutingGraphCPT**, che ha il compito di creare una sequenza di marker conforme alla soluzione del **Problema del Postino Cinese (CPT)**. **CPTRoutedSpawnArea** imposta se stessa come il nodo iniziale del grafo, fungendo da punto di partenza per il percorso degli agenti.

### 7.2.2 Parametri della simulazione

Per posizionare la segnaletica, abbiamo utilizzato il modulo Signboard Grid Generator (Capitolo 4.5). I cartelli sono stati collocati a un’altezza di 2 metri, orientati verso ovest, e distanziati di 2 metri l’uno dall’altro. Gli agenti vengono introdotti nell’edificio con una frequenza di 0.1 Hz.

La frequenza di aggiornamento della simulazione è impostata a 10 Hz e la sua durata è di 15 minuti, a cui precedono 2 minuti di warm-up, durante i quali gli agenti esplorano l’edificio senza considerare la segnaletica. Questo periodo di warm-up è stato introdotto per evitare che i cartelli posti vicino all’area di spawn ricevano un numero elevato di visite, influenzando negativamente i risultati finali.

La risoluzione del *Visibility Handler* (Capitolo 4.4.1) è stata impostata a 5 punti per metro, e il gradiente dei colori nell’output indica con colori caldi i punti di maggiore visibilità e con colori freddi quelli meno visitati.

---

<sup>2</sup>Chinese Postman Tour

### 7.2.3 Risultati

Analogamente alla **Coverage**, è possibile esportare i dati relativi alla visibilità della segnaletica in formato **CSV**. Questi dati, normalizzati tra i vari piani, possono essere analizzati per i diversi tipi di agenti (Tabella 7.3 e Tabella 7.4).

Tabella 7.3: Visibility per *Adult - Male*

Name	Visibility[Adult - Male]
Signboard [18, 0](Decke-002)	9.990812
Signboard [18, 1](Decke-002)	9.403118
Signboard [19, 2](Decke-002)	8.815423
Signboard [20, 0](Decke-002)	8.815423
Signboard [19, 1](Decke-002)	8.815423
Signboard [20, 1](Decke-002)	8.815423
Signboard [20, 3](Decke-002)	8.227728
Signboard [20, 2](Decke-002)	8.227728
Signboard [8, 2](Decke-002)	7.640033
Signboard [17, 2](Decke-002)	7.640033
Signboard [18, 2](Decke-002)	7.640033
Signboard [14, 2](Decke-002)	7.052339
Signboard [19, 3](Decke-002)	7.052339
Signboard [6, 2](Decke-002)	6.464643
Signboard [5 2](Decke-002)	6.464643

Tabella 7.4: Visibility per *Wheelchair User*

Name	Visibility[Wheelchair User]
Signboard [18, 0](Decke-002)	10.387720
Signboard [20, 0](Decke-002)	9.776674
Signboard [20, 3](Decke-002)	9.776674
Signboard [19, 2](Decke-002)	9.165632
Signboard [19, 1](Decke-002)	9.165632
Signboard [20, 1](Decke-002)	9.165632
Signboard [18, 2](Decke-002)	9.165632
Signboard [20, 2](Decke-002)	8.554589
Signboard [6, 2](Decke-002)	8.554589
Signboard [17, 2](Decke-002)	7.943548
Signboard [18, 1](Decke-002)	7.943548
Signboard [17, 1](Decke-002)	7.943548
Signboard [8, 2](Decke-002)	7.943548
Signboard [19, 3](Decke-002)	7.332506
Signboard [16, 3](Decke-001)	6.721463

Dai dati grezzi emerge una significativa differenza tra i cartelli più visibili per i diversi tipi di agenti. Questo è principalmente dovuto al fatto che la segnaletica è stata posizionata a un'altezza di 2 metri, una posizione non ideale per gli utenti in sedia a rotelle.

Un aspetto più interessante è la possibilità di visualizzare i dati della visibilità come texture applicate ai vari **Visibility Plane** (Figura 7.4 e Figura 7.5).

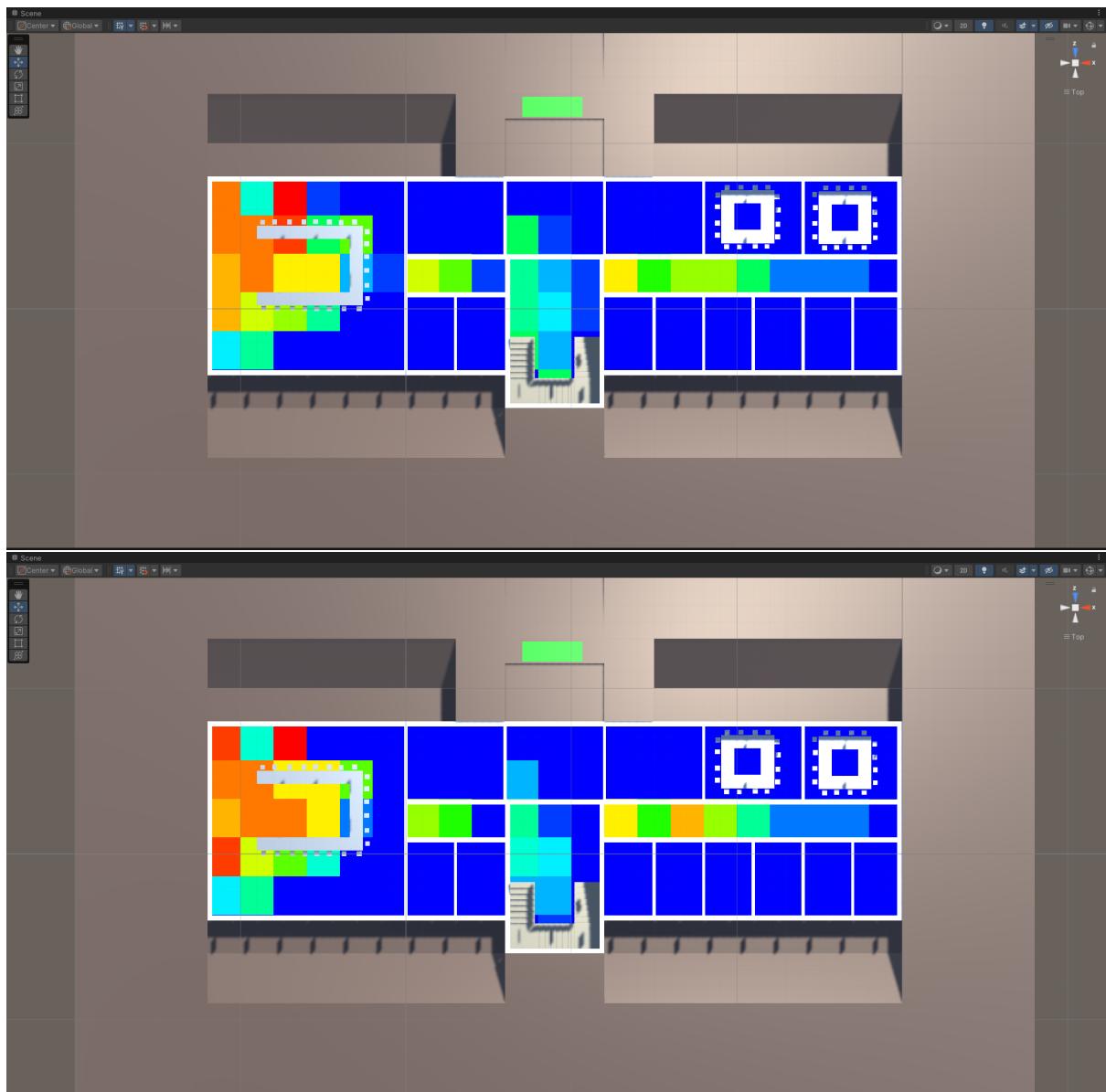


Figura 7.4: Piano Interrato. Risultati per *Adult - Male* in alto e *Wheelchair User* in basso.

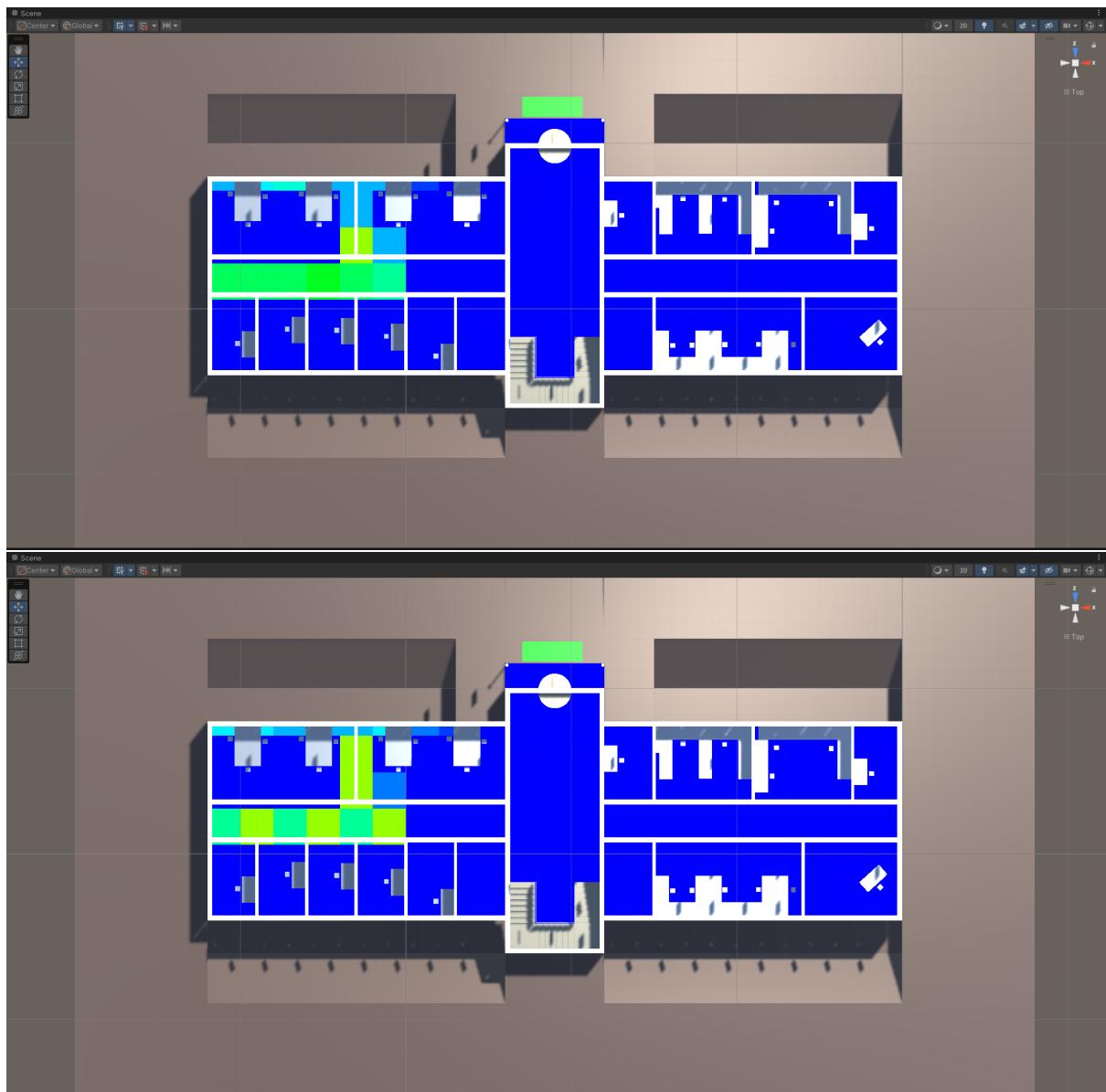


Figura 7.5: Piano Terra. Risultati per *Adult - Male* in alto e *Wheelchair User* in basso.

Questa modalità di visualizzazione evidenzia i cosiddetti **hotspot**, ovvero le aree dove la segnaletica è stata notata con maggiore frequenza. Tali informazioni possono essere utilizzate per decidere dove collocare cartelli importanti, massimizzando la loro visibilità.

# Capitolo 8

## Conclusioni

Il progetto presentato ha affrontato in modo efficace il problema della visibilità e della leggibilità della segnaletica all'interno degli edifici. Attraverso l'integrazione di modelli BIM con un sistema di simulazione multi-agente, è stato possibile costruire un ambiente virtuale che permette di analizzare come la segnaletica viene percepita da diverse categorie di individui, tenendo conto di fattori chiave come la geometria dell'edificio, gli ostacoli presenti e il flusso pedonale.

Uno degli aspetti più rilevanti del sistema sviluppato è la sua flessibilità. L'architettura modulare permette di definire agenti con diversi comportamenti e di adattare i parametri in funzione dell'edificio analizzato. Questo rende lo strumento versatile e facilmente applicabile in una vasta gamma di contesti, rendendolo una risorsa utile per professionisti del settore, come architetti e ingegneri, che desiderano ottimizzare il posizionamento della segnaletica in ambienti complessi.

I risultati delle simulazioni mostrano come, anche in un ambiente strutturalmente complesso, l'utilizzo di una segnaletica ben posizionata possa influenzare positivamente il tempo di percorrenza e la percentuale di successo degli agenti nel raggiungere i loro obiettivi. La possibilità di identificare "hotspot" nei percorsi pedonali e di ottimizzare la disposizione dei cartelli ha dimostrato di essere un approccio efficace per migliorare l'orientamento degli utenti.

L'analisi comparativa tra i diversi tipi di agenti (Perfect Agent, Wanderer Agent, SignBlind Agent) ha evidenziato che, sebbene l'effetto della segnaletica non sia sempre evidente in termini di riduzione della lunghezza del percorso, essa ha un impatto significativo sulla capacità degli agenti di raggiungere i loro obiettivi in modo efficiente. In particolare, Wanderer Agent, che si basa sulle informazioni fornite dalla segnaletica, ha mostrato prestazioni superiori rispetto agli agenti che ignorano completamente i cartelli. Data la complessità dell'agente in questione, esso richiederà ancora del lavoro perché

produca dei risultati affidabili, ma una volta migliorato diventerà uno strumento fondamentale per il confronto dell'efficacia di diverse configurazione di cartelli.

La scelta di non affrontare scenari di emergenza, concentrandosi invece sulla gestione ordinaria degli spazi, ha permesso di mantenere il progetto focalizzato su situazioni di navigazione quotidiana. Questa decisione progettuale è stata presa per circoscrivere il problema a un contesto più gestibile, evitando la complessità aggiuntiva che gli scenari di emergenza comportano, e permettendo così una modellazione più specifica e adattabile a situazioni ordinarie.

Il sistema sviluppato ha dimostrato di essere uno strumento flessibile e ben adattabile a contesti diversi, fornendo un valido supporto per l'analisi della visibilità della segnaletica. Le simulazioni hanno confermato che, attraverso un'attenta modellazione del flusso pedonale e dell'ambiente, è possibile ottenere dati utili per migliorare l'efficacia della segnaletica negli edifici.

## 8.1 Sviluppi futuri

Il progetto presentato offre una solida base per l'analisi della segnaletica in ambienti complessi, ma esistono numerose possibilità di ampliamento che potrebbero migliorarne l'efficacia e la versatilità.

Uno spunto interessante proviene dal lavoro di Mizar Luca Federici, Lorenza Manenti e Sara Manzoni, intitolato "*A Checklist for the Evaluation of Pedestrian Simulation Software Functionalities*" [14], che suggerisce varie funzionalità utili per migliorare i software di simulazione pedonale. Alcuni di questi aspetti sono già stati inclusi nel progetto, ma esistono ulteriori funzionalità che potrebbero essere introdotte:

- **Integrazione di formati CAD 2D:** Estendere lo strumento per supportare formati bidimensionali come .dxf o .dwg, permetterebbe una maggiore flessibilità nell'importazione di modelli architettonici.
- **Percorsi e comportamenti avanzati:** Migliorare la definizione di percorsi personalizzati e includere comportamenti avanzati, come fermate temporanee o attese, per simulare situazioni più complesse. L'aggiunta di aree che attraggono gli agenti consentirebbe inoltre di identificare zone di maggiore affluenza.
- **Configurazioni esportabili:** Implementare la possibilità di salvare ed esportare le configurazioni dei moduli utilizzati, facilitando la condivisione dei setup tra diversi utenti e team.

- **Strumenti di misura nel BIM:** L'integrazione di strumenti di misura direttamente all'interno del modello BIM semplificherebbe il flusso di lavoro, eliminando la necessità di passare tra software diversi.
- **Output di statistiche avanzate:** Fornire statistiche più dettagliate, come la densità dei pedoni o il tempo di permanenza degli agenti, arricchirebbe l'analisi dei flussi pedonali.

Attualmente, il progetto si basa sul formato IFC per l'importazione dei modelli BIM, ma esplorare l'adozione del formato **dotbim**, più leggero e facile da gestire, potrebbe rappresentare un miglioramento in termini di prestazioni e semplicità d'uso.

Un'altra possibile direzione riguarda l'ottimizzazione dei parametri associati al **DecisionNodeState**. Al momento, le decisioni degli agenti sono influenzate da pesi che potrebbero essere ottimizzati tramite uno studio sistematico, utilizzando approcci di tipo *brute-force* per identificare combinazioni ottimali che minimizzano il tempo di percorrenza o massimizzano la probabilità di successo nel raggiungimento degli obiettivi.

Inoltre, gli agenti attualmente non sono in grado di utilizzare gli ascensori, ma questa limitazione potrebbe essere facilmente risolta. L'identificazione degli ascensori tramite il tag IFC è già possibile e potrebbe essere implementata utilizzando il modulo **NavMeshLink**, che permette di collegare superfici NavMesh separate.

Un possibile area di indagine è lo studio del sistema di marker attualmente implementato, che utilizza le porte e le scale come punti principali di riferimento. Sebbene questa strategia si sia dimostrata efficace, potrebbe non essere sufficiente in tutti i contesti. Sarebbe utile uno studio specifico per verificare se l'uso esclusivo di porte e scale sia sufficiente a coprire la totalità dell'edificio o se vi siano situazioni in cui siano necessari marker aggiuntivi, come corridoi particolarmente lunghi o aree con poche aperture.

Lo strumento, attualmente, dipende dall'editor di Unity per tutte le operazioni. Per diventare una soluzione completa e autonoma, è necessaria l'introduzione di una serie di strumenti, tra una interfaccia grafica (GUI).

Le funzionalità principali potrebbero includere:

- Movimento libero della telecamera.
- Possibilità di nascondere parti del modello 3D per una visualizzazione più chiara.
- Aggiunta e modifica di oggetti direttamente dall'interfaccia grafica, come le *Spawn Area* e la segnaletica.

## **Estensione a scenari di emergenza**

Inizialmente era stata considerata l'implementazione di modelli per scenari di emergenza, utilizzando approcci complessi come quello proposto da Crociani et al. [4]. Tuttavia, per evitare di complicare eccessivamente il progetto e mantenere l'attenzione sulla segnaletica informativa, questa funzionalità è stata esclusa. In futuro, sarebbe possibile introdurre scenari di emergenza, ampliando lo strumento per analizzare l'efficacia della segnaletica di evacuazione e ottimizzare le dinamiche di fuga durante le emergenze.

# Bibliografia

- [1] ACCA Software. *IFC schema part 3: The IFCPropertyDefinition*. Accessed: 23-10-2024. 2020. URL: <https://biblus.accasoftware.com/en/ifc-schema-part-3-the-ifcpropertydefinition/>.
- [2] Peter Barnes e Nigel Davies. *BIM in Principle and in Practice*. 2014.
- [3] Christian Becker-Asano. *Multi-Agent System with Unity3D for airport passenger simulation*. Youtube. 2013. URL: <https://youtu.be/uVpN136q7N8>.
- [4] Luca Crociani et al. «Route choice in pedestrian simulation: Design and evaluation of a model based on empirical observations». In: *Intelligenza Artificiale* 10 (2016). 2, pp. 163–182. ISSN: 2211-0097. DOI: 10.3233/IA-160102. URL: <https://doi.org/10.3233/IA-160102>.
- [5] Rohit K Dubey et al. «Information Theoretic Model to Simulate Agent-Signage Interaction for Wayfinding». In: *Cognitive Computation* (1 gen. 2019), pp. 1–18. published.
- [6] C.M. Eastman. *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. Wiley, 2011. ISBN: 9780470541371. URL: <https://books.google.it/books?id=-GjrBgAAQBAJ>.
- [7] Lazaros Filippidis et al. «Representing the influence of signage on evacuation behavior within an evacuation model». In: *Journal of Fire Protection Engineering* 16.1 (2006), pp. 37–73.
- [8] Peter E. Hart, Nils J. Nilsson e Bertram Raphael. «A Formal Basis for the Heuristic Determination of Minimum Cost Paths». In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.
- [9] Dirk Helbing e Péter Molnár. «Social force model for pedestrian dynamics». In: *Phys. Rev. E* 51 (5 1995), pp. 4282–4286. DOI: 10.1103/PhysRevE.51.4282. URL: <https://link.aps.org/doi/10.1103/PhysRevE.51.4282>.
- [10] <https://www.steptools.com>. URL: <https://www.steptools.com/docs/stpfiles/ifc/>.

- [11] *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries*. International Standard ISO 16739-1:2024. Geneva, Switzerland: International Organization for Standardization, 2024. URL: <https://www.iso.org/standard/84123.html>.
- [12] Pablo Marin-Plaza et al. «Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles». In: *Journal of Advanced Transportation* 2018 (feb. 2018), p. 6392697. ISSN: 0197-6729. DOI: 10.1155/2018/6392697. URL: <https://doi.org/10.1155/2018/6392697>.
- [13] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. 1st. USA: Prentice Hall Press, 2017. ISBN: 0134494164.
- [14] Sara Manzoni Mizar Luca Federici Lorenza Manenti. *A Checklist for the Evaluation of Pedestrian Simulation Software Functionalities*. Rapp. tecn. 2014.
- [15] Ali Motamedi et al. «Signage visibility analysis and optimization system using BIM-enabled virtual reality (VR) environments». In: *Advanced Engineering Informatics* 32 (2017), pp. 248–262. DOI: <https://doi.org/10.1016/j.aei.2017.03.005>. URL: <https://www.sciencedirect.com/science/article/pii/S1474034616301203>.
- [16] Mehdi Moussaïd et al. *Experimental study of the behavioural mechanisms underlying self-organization in human crowds*. Rapp. tecn. 2009. DOI: <https://doi.org/10.48550/arXiv.0908.3131>. URL: <https://royalsocietypublishing.org/doi/abs/10.1098/rspb.2009.0405>.
- [17] Thomas Paviot et al. «STEP and IFC export to X3D». In: *Proceedings of the 25th International Conference on 3D Web Technology* (2020). URL: <https://api.semanticscholar.org/CorpusID:226283054>.
- [18] ACCA Software. *IFC file*. White Paper. URL: [https://bim.acca.it/wp-content/uploads/2021/04/IFC-file\\_Whitepaper.pdf](https://bim.acca.it/wp-content/uploads/2021/04/IFC-file_Whitepaper.pdf).
- [19] Harold Thimbleby. «The directed Chinese Postman Problem». In: *Software: Practice and Experience* 33.11 (2003), pp. 1081–1096. DOI: <https://doi.org/10.1002/spe.540>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.540>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.540>.
- [20] *Unity - Manual: Creating a NavMesh Agent*. Unity Technologies. 2023. URL: <https://docs.unity3d.com/Manual/nav>CreateNavMeshAgent.html>.

# Elenco delle tabelle

7.1	Coverage per <i>Adult - Male</i> . . . . .	67
7.2	Coverage per <i>Wheelchair User</i> . . . . .	68
7.3	Visibility per <i>Adult - Male</i> . . . . .	72
7.4	Visibility per <i>Wheelchair User</i> . . . . .	73

# Elenco delle figure

2.1	Esempio gerarchia degli elementi in un file IFC. Fonte: [17] . . . . .	12
2.2	Esempio di proprietà assegnabili ad un entità nello schema IFC. Fonte: [1]	13
3.1	Esempio di <i>VCA</i> in blu. Fonte: [15] . . . . .	16
3.2	Esempio di <i>Navigation Mesh</i> che si estende su due livelli. Fonte: [20] . .	18
3.3	Test 1: Gruppi di agenti attraversano un corridoio in direzione opposta. <a href="https://www.youtube.com/watch?v=08RZ2dKRbTs">https://www.youtube.com/watch?v=08RZ2dKRbTs</a> . . . . .	20
3.4	Test 2: Da 10 a 100 agenti che passano attraverso una strettoia. <a href="https://www.youtube.com/watch?v=0ackMvqqVa8">https://www.youtube.com/watch?v=0ackMvqqVa8</a> . . . . .	21
3.5	Test 3: Due gruppi di 1000 agenti attraversano un corridoio in direzione opposta. Questo test rappresenta condizioni estreme, ma rende molto evidenti i limiti dello strumento fornito da Unity. <a href="https://www.youtube.com/watch?v=0ackMvqqVa8">https://www.youtube.com/watch?v=0ackMvqqVa8</a> . . . . .	22
4.1	Come si presenta l'Editor di Unity quando viene creato un nuovo progetto	23
4.2	Diagramma UML delle classi usate per immagazzinare i dati provenienti dall' <i>IFC Schema</i> . . . . .	25
4.3	Interfaccia del modulo IfcOpenShellParser . . . . .	26
4.4	Come si presenta la NavMesh (in azzurro) di una sezione di un edificio. .	27
4.5	Parametri del <i>NavMesh Agent</i> . . . . .	29
4.6	Esempio di <i>IfcSlab</i> con evidenziati i triangoli che compongono la mesh .	31
4.7	Area di visibilità di un cartello. Fonte: [15] . . . . .	32
4.8	Un <i>Visibility Plane</i> con evidenziati in blu i punti che compongono la griglia, con una risoluzione di 5 punti/metro . . . . .	33
4.9	Esempio di <i>area di visibilità</i> (blu), evidenziata all'interno di un <i>piano di visibilità</i> (rosso). Fonte: [15] . . . . .	34
4.10	Interfaccia del modulo Visibility Handler . . . . .	35
4.11	Interfaccia del modulo Visibility Plane Generator . . . . .	35
4.12	Piano di Visibilità del primo piano di un edificio. Da notare come il piano è sollevato all'altezza degli occhi di un agente (1.75m) ed i colori delle VCA corrispondono ai colori dei cartelli. . . . .	36
4.13	Interfaccia del modulo Signboard Grid Generator . . . . .	37
4.14	Interfaccia del modulo Marker Generator. . . . .	39
4.15	Esempio di marker (in blu) in una sezione di un edificio. . . . .	40

5.1	Diagramma delle dipendenze tra i vari componenti degli agenti. . . . .	42
5.2	Rappresentazione grafica della funzione di Raycast. In verde i raggi che vanno alla ricerca del muro ed in rosso le collisioni. . . . .	46
5.3	Macchina a stati finiti di Wanderer Agent. . . . .	51
5.4	Esempio di Spawn Area (in verde) posizionata all'ingresso di un edificio.	54
5.5	Esempio di prefabbricato di un agente ( <i>Agent Wanderer</i> ). A destra è visibile l'elenco dei moduli ad esso collegati. . . . .	55
5.6	Interfaccia di GoalAgentSpawnArea. . . . .	56
6.1	Rappresentazione tridimensionale del corridoio usato per i test. . . . .	59
6.2	Risultati originali di Mehdi Moussaïd et al. [16]. In alto: Test in condizione 2. In basso: Test in condizione 3. . . . .	60
6.3	Risultati della nostra implementazione. In alto: Test in condizione 2. In basso: Test in condizione 3. . . . .	60
6.4	Macchina a stati finiti di SignBlind Agent. . . . .	62
6.5	Confronto delle performance tra i tre tipi di agente. I alto il test che mette a confronto il tempo impiegato, al centro la lunghezza del percorso ed in basso la percentuale di successi. . . . .	63
7.1	Come si presenta il modello 3D del <i>Karlsruhe Institute of Technology</i> all'interno dell'editor di Unity. . . . .	65
7.2	Barplot della coverage totale delle prime 15 <i>Signboard</i> con coverage più alta. . . . .	68
7.3	Rappresentazione grafica degli archi del grafo che collegano i vari marker. I muri esterni sono stati nascosti per migliorare la visibilità dei nodi. . . . .	70
7.4	Piano Interrato. Risultati per <i>Adult - Male</i> in alto e <i>Wheelchair User</i> in basso. . . . .	74
7.5	Piano Terra. Risultati per <i>Adult - Male</i> in alto e <i>Wheelchair User</i> in basso. . . . .	75

# Ringraziamenti

Desidero esprimere la mia gratitudine a tutti coloro che mi hanno accompagnato durante il percorso universitario, rendendolo più ricco, anche con piccoli gesti.

Un ringraziamento va al professor Giuseppe Vizzari, relatore di questo lavoro, per i suggerimenti e il supporto forniti durante lo sviluppo del progetto.

Un sentito grazie alla mia amica e compagna di viaggio Lisa, che non solo ha condìvisio con me lezioni ed esami, ma ha anche contribuito allo sviluppo del progetto.

Un ringraziamento speciale ai miei genitori, che mi hanno sostenuto sia moralmente che finanziariamente, rendendo possibile il raggiungimento di questo importante obiettivo.

Infine, un ringraziamento a mio fratello, geometra, che grazie alla sua professione è stato in grado di darmi preziosi consigli.