ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica - Scienza ed Ingegneria
Corso di Laurea Magistrale in Informatica

Simulazione della dinamica del traffico in Computer Graphics

Tesi di Laurea Magistrale in Informatica

Relatore:
Prof.ssa **Serena Morigi**

Presentata da:
Leonardo Tamai

Co-Relatore:
Prof. **Daniele Federico**

Sessione Dicembre 2024
Anno Accademico 2023-2024

Indice

Abstract

L'obiettivo del lavoro coperto da questa tesi è duplice: approfondire quella che in Computer Graphics è la tematica della gestione di folle di agenti in ambienti tridimensionali, e applicare queste conoscenze per estendere il plugin Atoms per Autodesk Maya , sviluppato dall'azienda Tool Chefs, per permettere di creare nel più chiaro, ma dettagliato modo possibile, una scena urbana realistica, adattabile a contesti cinematografici, videoludici o simulativi. La tesi esplora in dettaglio le fasi di input dei dati da parte dell'utente, la costruzione di oggetti fondamentali e degli agenti, e l'implementazione della loro logica comportamentale, attività integrate da solide basi teoriche di matematica e Computer Graphics.

Glossario

Locator: nodo di Maya adibito alla visualizzazione di semplici geometrie/forme in Maya. Equivalente ai markers di Blender.
Traffic locator: un nodo contenente le informazioni globali per la gestione del traffico.
Viewport: nei software 3D come Autodesk Maya, la viewport è la finestra principale dell'interfaccia in cui vengono visualizzati e manipolati gli oggetti della scena.
Crossing locator: un elemento 3D all'interno della viewport utile per la visualizzazione di un incrocio.
Street locator: un elemento 3D all'interno della viewport utile per la visualizzazione di una strada e il suo flusso.
Agent/Agente: una singola entità di una crowd (folla), in questo caso un veicolo.
Agent Type: un tipo di agente, si può considerare un agent type come la classe di un agente, mentre l'agente stesso è l'istanza di un agent type.
Metadata: dati contenuti da ogni agente e utilizzati da Atoms durante la simulazione. Per esempio, alcuni metadata sono "position" e (la posizione dell'agente), "direction" (la direzione dell'agente).

Capitolo 1

Introduzione

Lo scopo di questa tesi è analizzare, dal punto di vista della Computer Graphics, come avviene la creazione, animazione e la programmazione di scene 3D complesse, ed in particolare della gestione delle folle, ovvero la branca degli scenari al cui interno sono presenti un grosso numero di agenti che, a seconda della tipologia di comportamento assegnatogli, interagiscono fra di loro o con l'ambiente che popolano. Per fare ciò, le nozioni teoriche verranno affiancate dalla descrizione dell'attività di tirocinio, nonchè progetto di tesi, svolta presso l'azienda londinese Tool Chefs, specializzata nella creazione di plugin unici per una varietà di software, come Autodesk Maya, The Foundry Nuke, SideFX Houdini e Newtek Lightwave, di applicazioni autonome e in servizi di consulenza. I loro strumenti sono principalmente mirati all'animazione, ma sono anche esperti in pipeline, rigging, modellazione, folla e illuminazione. Fra i vari strumenti messi a disposizione dall'azienda c'è Atoms, un framework crowd autonomo progettato per gestire grandi quantità di dati di animazione, integrato in software come Unreal Engine, Autodesk Maya e SideFX Houdini.

Figura 1.1: Logo di Tool Chefs (a destra) e logo di Atoms (a sinistra)

Il progetto trattato consiste nell'utilizzare il plugin Atoms e le sue librerie in Autodesk Maya per creare un'applicazione in grado di ricreare in modo più semplice ed intuitivo possibile uno scenario urbano realistico in 3D, composto da strade costituite da sensi di marcia, limiti di velocità, corsie, incroci in grado di gestire semafori e code, ed infine ovviamente i veicoli, agenti che seguono un certo numero di norme comportamentali come velocità, distanza di sicurezza, sorpassi e via dicendo. I requisiti di funzionamento e tecnologici del traffic behaviour di Atoms, nonché 3

principali componenti del sistema (che verranno approfonditi dettagliatamente nei capitoli successivi) sono:

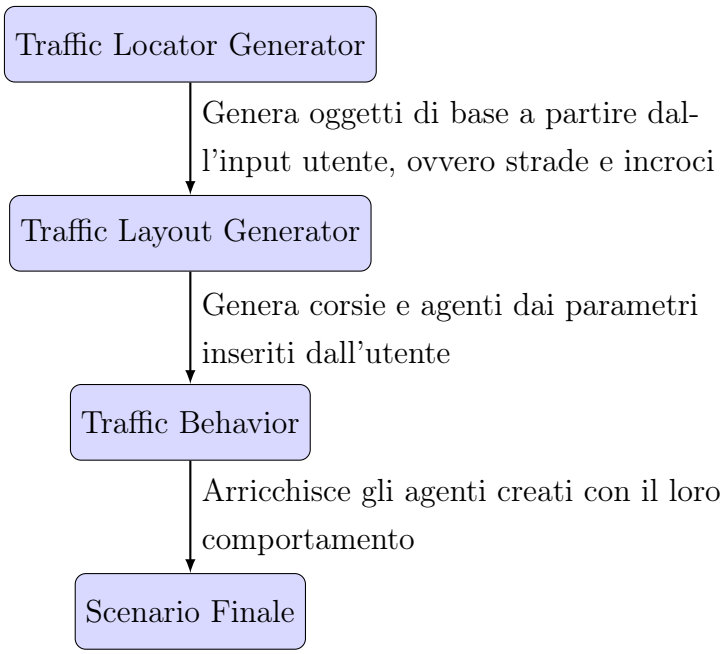
- Un generatore di *traffic locator* partendo da delle curve di animazione definite dall'utente.
- Un *traffic layout generator* per la generazione dei veicoli al primo frame della simulazione.
- Un *behaviour module* per la gestione del comportamento dei veicoli.

L'ultimo punto, ovvero il modulo per il comportamento degli agenti, è stato ultimato solamente in parte per motivi tempistici, fornendo quindi comportamenti basilari.
Nei capitoli successivi verranno quindi trattati gli elementi teorici fondamentali matematici e di Computer Graphics che riguardano l'animazione e la gestione di folle di agenti, portando, ove possibile, la loro applicazione pratica facendo riferimento all'attività di tirocinio sulla simulazione di uno scenario urbano scritto in C++ utilizzando il plugin Atoms per Autodesk Maya.

Capitolo 2

Componenti da sviluppare e obiettivi

Come anticipato precedentemente, l'interezza del progetto è suddividibile in tre componenti fondamentali, quali il Traffic Locator Generator, il quale comprende la creazione degli oggetti primordiali del sistema, come le strade e gli incroci, il Traffic Layout Generator, il cui compito è quello di generare gli agenti usufruendo dei parametri inseriti dall'utente, infine il Traffic Behavior, che si occupa di gestire il comportamento che gli agenti devono mantenere per la durata della simulazione. Di seguito uno schema a blocchi riassuntivo e nei prossimi capitoli un'analisi più dettagliata di questi aspetti fondamentali.



2.1 Strumenti utilizzati

- **Visual Studio**, IDE con cui è stata sviluppato il progetto;
- **Autodesk Maya** ed il plugin **Atoms**, software professionale di modellazione, animazione e rendering 3D;
- **Discord**, piattaforma di comunicazione;
- **GitHub**, per gestire e condividere il codice.

Il progetto è scritto quasi interamente in C++, ad eccezione di due script in linguaggio Python che verranno trattati nelle sezioni successive.

2.2 Traffic Locator Generator

Come prima cosa l'utente dovrà creare delle curve nella viewport di Maya, queste curve definiranno le strade e automaticamente anche gli incroci che comporranno lo scenario, questo perchè se le estremità di due curve sono abbastanza vicine, generano automaticamente un incrocio che le collegherà.
Per fare ciò, vengono eseguiti due script in Python:

- una volta pronte, l'utente seleziona le curve e lancia il primo script che genera un *traffic locator* (oggetto di Atoms che raggruppa altri oggetti come le strade, gli incroci e gli agenti) e uno *street locator*, quest'ultimo per ogni curva;
- successivamente l'utente seleziona il *traffic locator* oppure uno o più *street locators* e lancia il secondo script che genera i *crossing locators* nei punti necessari. Per punto necessario, si intende uno in cui il punto finale o iniziale di una curva è in prossimità al punto finale o iniziale di una o più curve. La funzione dovrà prendere come input un parametro *distance* che permetterà alla logica di questa funzione di utilizzare per ogni incrocio solo i punti la cui distanza è inferiore o uguale a questo valore.

Dall'interfaccia di Atoms l'utente potrà personalizzare i parametri che verranno utilizzati dai moduli successivi per generare lo scenario e gli agenti, parametri che verranno discussi nelle sezioni successive.

2.3 Traffic Layout Generator

Un layout generator è una classe di Atoms adibita alla generazione di agenti. In questo caso, il traffic layout generator dovrà creare veicoli lungo le strade definite nel passaggio precedente. Il generator verrà creato all'interno del behaviour module (prossimo step) e passato ad Atoms, il quale provvederà a generare gli agenti al primo frame della simulazione.

2.3.1 Location settings

Ogni locator sarà mappato da una classe di Atoms che sarà accessibile all'interno del traffic behaviour module. Il Behaviour module dovrà prendere in considerazione i seguenti parametri per la creazione del traffico, chiamati Global Traffic Settings (locator):

- Drive side (senso di marcia)
- Enable switch lane (per abilitare il cambiamento di corsia)
- Connections of street locators
- Connections of crossing locators

Attributi delle strade:

- lane switch duration (tempo per cambiare corsia)
- lane width (la larghezza di ciascuna corsia)
- lane separation (lo spazio tra ciascuna corsia)
- lane count 1 (il numero di corsie in un senso, se il numero è zero la strada è a senso unico nel verso opposto)
- lane count 2 (il numero di corsie nel senso opposto, se il numero è zero la strada è a senso unico nel verso opposto)
- speed limit (la velocità massima della strada)
- Allow U-Turn (inversione a U)

Attributi degli incroci:

- Crossing radius (quanto è grande il crossing)

- Enable light (se questo crossing è gestito da un semaforo)
- Red timer (quanto dura la luce rossa)
- Green timer (quanto dura la luce verde)
- Orange timer (quanto dura la luce arancione)

2.4 Traffic Behavior

Il traffic behaviour si occuperà di preparare il traffic layout generator, passarlo ad Atoms al primo frame della simulazione ed infine gestire il movimento delle macchine all'interno delle strade e crossings. Per la gestione dei veicoli, sarà necessario modificare alcuni dati (metadata) di ogni agente:

- direction (il vettore direzionale dell'agente);
- position (posizione dell'agente);
- World matrix dei veicoli (una matrice di trasformazione che rappresenta la posizione, l'orientamento e la scala di ogni agente all'interno del sistema di coordinate globali).

A ciascun veicolo all'inizio della simulazione verrà assegnato un punto nel traffic system e cercherà di raggiungerlo.

2.4.1 Behaviour module settings

Il Behaviour module dovrà includere e prendere in considerazione i seguenti parametri per la gestione del traffico.

- velocità minima e massima;
- velocità di inversione minima e massima;
- ignoranento del limite di velocità;
- distanza di sicurezza minima e massima;
- accelerazione minima e massima;
- modalità di navigazione, tra cui: punto di destinazione casuale, punto di destinazione indicato, sequenza di punti di destinazione, metadata (per es. un altro agente);

- sfasamento minimo e massimo (parametri per offset sulla normale delle curve per fare in modo che la macchina non sia proprio al centro della lane, usato per la creazione degli agenti);
- massimo numero di agenti (usato per la creazione degli agenti);
- numero di iterazioni (usato per la creazione degli agenti);
- distanza minima e massima (usato per la creazione degli agenti).

2.5 Obiettivi

Lo scopo del progetto è realizzare uno strumento per Maya in grado di creare, con il miglior compromesso possibile fra chiarezza e livello di dettaglio, uno scenario urbano popolato da veicoli il più vicino possibile alla realtà, volto all'utilizzo in ambito cinamatografico, videoludico e simulativo.

Capitolo 3

Animazione delle folle e stato dell'arte

I modelli di agenti di massa (o crowd simulation models) sono tecniche usate per simulare il comportamento di un grande numero di agenti autonomi (come persone, creature o veicoli) che interagiscono tra loro in un ambiente condiviso. Questi modelli vengono applicati sia nei film che nei videogiochi per creare movimenti realistici di folle o di grandi gruppi di personaggi.

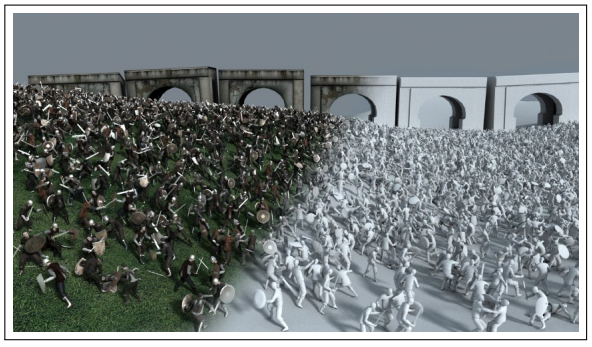


Figura 3.1: Esempio di simulazione di una folla usando il software MASSIVE (Multiple Agent Simulation System in Virtual Environment)

Nelle sezioni di questo capitolo verranno analizzati dal punto di vista prettamente teorico l'animazione delle folle e sistemi che le caratterizzano, dando un'ampia occhiata allo stato dell'arte.

3.1 Sistemi stocastici

Nelle simulazioni di agenti, specialmente in contesti complessi come il traffico urbano o l'animazione di folle, i sistemi stocastici sono utilizzati per introdurre variabilità

e realismo nei comportamenti degli agenti. Tali sistemi utilizzano numeri casuali per determinare azioni e decisioni degli agenti, ma spesso consentono anche di usare seed randomici personalizzabili per ottenere simulazioni riproducibili.
Un sistema stocastico è un modello in cui alcuni processi sono determinati da variabili casuali. In altre parole, il comportamento del sistema non è deterministico, ma si basa su probabilità e distribuzioni casuali. Questo è fondamentale per gestire agenti in simulazioni complesse, dove è importante che il comportamento non sia completamente prevedibile. La variabilità nella velocità di guida, nelle decisioni di sorpasso, nella scelta del percorso e nel tempo di reazione ai semafori può essere gestita tramite variabili casuali.
I seed randomici (semi casuali) sono utilizzati per inizializzare i generatori di numeri casuali in modo che producano sempre la stessa sequenza di numeri, dato un certo seed. Questo è utile quando si vuole poter riprodurre una simulazione esatta più volte, nonostante l'uso di processi casuali. Questi hanno due grossissimi vantaggi:

- Riproducibilità: Se una simulazione stocastica dà risultati desiderati, è possibile "fissare" il seed randomico per riprodurre esattamente gli stessi risultati in esecuzioni successive.

- Debugging: Per simulazioni complesse, è utile avere la capacità di ripetere una simulazione stocastica, per poter individuare e risolvere eventuali problemi.

Nel caso di Atoms, coperto dal progetto, viene fornita una classe di nome `Rand48`, che rappresenta un generatore di numeri pseudo-casuali uniformemente distribuito di 48 bit, basato sul valore iniziale con cui viene costruito (seed), viene utilizzato per generare sequenze riproducibili di numeri casuali. Può restituire numeri casuali in vari formati e intervalli. Può restituire sequenze di variabili booleane, intere oppure in virgola mobile, ed al suo interno contiene un array di tre dimensioni per memorizzare lo steto del generatore; la sua ultima funzione è generare un valore in virgola mobile contenuto all'interno di un intervallo di cui si specificano gli estremi.

3.2 Algoritmi di steering (guidance)

Gli algoritmi di steering (guidance) sono un insieme di tecniche utilizzate per simulare il movimento di agenti autonomi, come personaggi o veicoli, in ambienti virtuali. Sono particolarmente utili in ambiti come l'animazione, i videogiochi e le simulazioni di folle, dove è necessario che gli agenti si muovano in modo realistico, evitando collisioni e interagendo con l'ambiente circostante. Il termine "steering"

significa letteralmente "guidare", e gli algoritmi in questione guidano il comportamento dinamico degli agenti per spostarli da un punto all'altro, seguendo regole specifiche.
Questi algoritmi sono stati introdotti da Craig Reynolds nel 1987 con il suo lavoro sui "Boids", un sistema che simula il comportamento collettivo di stormi di uccelli, e si basano su forze vettoriali che determinano la direzione e l'accelerazione di un agente. Ogni agente è rappresentato da una posizione (spazio 2D o 3D), una velocità e una direzione. Gli algoritmi di steering generano forze che modificano la velocità e la direzione dell'agente, creando movimenti realistici che evitano ostacoli, seguono traiettorie e interagiscono con altri agenti.
I comportmenti di Steering principali sono:

- Seek (cerca): l'agente si muove verso un obiettivo o una posizione specifica. Questo comportamento genera una forza che spinge l'agente nella direzione del bersaglio, accelerando verso di esso fino a raggiungerlo. È comunemente utilizzato per far sì che gli agenti seguano un percorso o si dirigano verso un punto di interesse.

- Flee (Fuggi): È l'opposto di seek. L'agente si allontana da un determinato obiettivo o minaccia. Questo è utilizzato quando l'agente deve evitare il pericolo o una minaccia. È utile per simulare la fuga da nemici o oggetti pericolosi.

- Arrival (Arrivo): Simile a "seek", ma l'agente rallenta quando si avvicina all'obiettivo, per evitare di superarlo o fermarsi bruscamente. La velocità diminuisce man mano che l'agente si avvicina al bersaglio. È utilizzato per creare arrivi graduali, evitando che gli agenti si muovano in modo innaturale o si fermino improvvisamente.

- Wander (Vagare): L'agente si muove in modo casuale nell'ambiente, cambiando direzione in modo imprevedibile. Questo comportamento crea movimenti naturali e caotici, come quelli di una creatura che esplora un'area senza una meta precisa. Utilizzato per simulare comportamenti erratici o di esplorazione in ambienti aperti.

- Pursue (Insegui): L'agente cerca di intercettare un bersaglio in movimento, predicendo dove si troverà il bersaglio in futuro in base alla sua velocità e direzione attuali. Questo crea una simulazione più realistica rispetto a semplicemente cercare di raggiungere il bersaglio alla sua posizione attuale. Utilizzato

in simulazioni di inseguimento, ad esempio per i predatori che inseguono una preda o per i veicoli che inseguono un bersaglio mobile.

- Evade (Evita): Simile a "flee", ma l'agente cerca di evitare un bersaglio mobile, anticipando i movimenti futuri dell'inseguitore. Questo è particolarmente utile in situazioni di difesa o fuga da una minaccia in movimento. Utilizzato per simulare personaggi che fuggono da nemici, veicoli o oggetti pericolosi che si muovono verso di loro.

- Obstacle Avoidance (Evitare gli ostacoli): Gli agenti devono evitare collisioni con ostacoli presenti nell'ambiente, cambiando direzione prima di colpire un oggetto. Essenziale per far sì che gli agenti si muovano in ambienti affollati o con ostacoli, senza camminare attraverso oggetti o scontrarsi tra loro.

- Separation (Seprarazione): L'agente si allontana dagli altri agenti vicini per evitare collisioni o sovraffollamento. Questo comportamento è fondamentale nelle simulazioni di folle o di stormi, dove ogni agente deve mantenere una certa distanza dagli altri. Utile nelle simulazioni di folle, stormi di uccelli, banchi di pesci o veicoli in un convoglio.

3.3 Gestione del traffico

Nella simulazione di un ambiente urbano di traffico in cui gli agenti sono automobili, non possono mancare dei concetti chiavi e tecniche, quali:

- Pathfinding (Ricerca del percorso): Gli agenti (le automobili) devono trovare il percorso ottimale per raggiungere una destinazione. Un approccio comune è l'algoritmo A* (A-star), che permette di calcolare il percorso più breve considerando una mappa con strade, incroci e ostacoli. Ogni auto calcola il proprio percorso verso la destinazione successiva e aggiorna il tragitto se la destinazione cambia.

- Algoritmi di Steering per Evitare Collisioni: Gli algoritmi di steering sono utilizzati per far sì che le automobili evitino collisioni tra di loro e con altri ostacoli. Gli agenti utilizzano comportamenti come evade, obstacle avoidance e separation. Le auto rallentano o cambiano corsia quando si avvicinano troppo ad altri veicoli o ostacoli, garantendo un movimento fluido e sicuro.

- Regole di Traffico: Le auto devono rispettare i semafori, le precedenze e le regole agli incroci. Le regole di traffico vengono implementate come parte delle

decisioni di percorso e di comportamento degli agenti. Ogni incrocio è un nodo in cui le auto devono fermarsi, attendere il verde o dare precedenza. Viene utilizzato un timer per i semafori e le auto valutano se fermarsi o avanzare. Quando molte auto si fermano a un semaforo, viene utilizzato un algoritmo di coda per distribuire le auto nei diversi flussi e impedire congestioni.

- Modelli di Flusso di Veicoli (Traffic Flow Models): Modello di Follow the Leader, è un modello dinamico in cui le auto seguono il veicolo davanti a loro, regolando velocità e distanza per evitare collisioni. Questo si ispira al comportamento umano di guida e simula movimenti su strade con diverse velocità.

- Aggiornamento della Destinazione: Ogni auto può avere una destinazione assegnata che cambia una volta raggiunta (ad esempio, auto di consegna, taxi, trasporto pubblico). Una volta raggiunta la destinazione, viene assegnato un nuovo obiettivo basato su una logica predefinita.

- Overtaking (Sorpassi): I sorpassi avvengono quando un'auto più veloce deve superare un veicolo più lento sulla stessa corsia. Gli agenti utilizzano un algoritmo che permette di cambiare corsia e sorpassare il veicolo più lento.

  - Lane Switching (Cambio di Corsia): Le auto valutano se è sicuro cambiare corsia, controllando la presenza di altri veicoli nelle corsie adiacenti.
  - Overtaking Conditions (Condizioni di Sorpasso): Se la velocità del veicolo di fronte è inferiore a una soglia prestabilita, l'agente valuta se cambiare corsia e accelerare.

- Velocità e Accelerazione: Gli agenti devono accelerare o frenare in modo realistico. Questo richiede modelli che tengano conto della fisica di accelerazione e decelerazione, nonché dei limiti di velocità.

- Interazione con l'Ambiente (Semafori, Stop, Incroci): Per i semafori, i veicoli si fermano quando il semaforo è rosso e procedono con il verde. Il comportamento viene gestito attraverso un algoritmo che tiene conto del colore del semaforo e del tempo di attesa. Inotre, gli agenti si fermano agli stop o rispettano le precedenze agli incroci. Le decisioni di movimento sono basate su regole predefinite che regolano chi ha la priorità all'incrocio.

3.4 Stato dell'arte

In questa sezione verranno evidenziati quelli che sono gli attuali impieghi, potenzialità e limiti della simulazione delle folle.

3.4.1 Applicazioni principali

Un largo successo lo ha avuto indubbiamente nel campo artistico come **cinema e videogiochi**, infatti nel capolavoro della trilogia cinematografica de Il Signore degli Anelli ha rivoluzionato il cinema grazie all'uso del software MASSIVE (Multiple Agent Simulation System in Virtual Environment). Sviluppato da Stephen Regelous per Wētā FX, MASSIVE è stato progettato per gestire folle di migliaia di personaggi autonomi, ciascuno dotato di intelligenza artificiale e comportamento individuale. Ogni agente aveva capacità sensoriali (visione, suoni) e regole di comportamento definite tramite *fuzzy logic* (per gestire concetti che non possono essere descritti rigidamente come veri o falsi), garantendo realismo anche in scene caotiche. Ad esempio, soldati che scappavano dal campo di battaglia venivano interpretati come vigliacchi, ma in realtà era un comportamento emergente naturale dovuto alle regole incomplete inserite nella loro "mente". MASSIVE è stato usato anche in film successivi come Avatar e King Kong per gestire scene complesse di folle o creature. In ambito videoludico, invece, la simulazione di folle è cruciale per creare ambienti immersivi, per esempio la famosa serie Assassin's Creed utilizza: *NavMesh (Navigation Meshes)* per consentire movimenti realistici, *LOD (Level of Detail)* per ridurre la complessità visiva e computazionale delle folle lontane ed *animazioni procedurali* per variare il comportamento dei personaggi non giocanti (NPC), migliorando il realismo.

Figura 3.2: Frame tratti dal film de Il Signore degli Anelli (destra) e dal videogioco Assassin's Creed Unity (sinistra)

Nel campo della **simulazione di emergenza** vengono utilizzati modelli come il *Social Force Model*, che trattano gli individui come particelle soggette a forze di attrazione (verso le uscite) e repulsione (da ostacoli o altre persone), anche modelli

agent-based sono utilizzati per simulare decisioni individuali, come seguire familiari o scegliere percorsi alternativi. Trovano applicazione della progettazione di edifici pubblici (ospedali, stadi) per ottimizzare le vie di fuga e nell'analisi di scenari di panico per eventi critici come incendi o esplosioni.

Figura 3.3: Simulazione di un'evacuazione con ostacoli in Unity3D

La simulazione delle folle viene usata nella **progettazione di infrastrutture urbane** per ottimizzare il movimento delle persone, ad esempio nella modellazione di flussi pedonali nei centri commerciali e stazioni ferroviarie o nei test virtuali di progetti per piazze, parchi o centri città prima della loro costruzione. Comuni tecnologie utilizzate sono le *microsimulazioni* per analizzare il comportamento individuale e i *lussi continui* per simulare movimenti densi e prevedere congestioni.

Figura 3.4: Simulazione della congestione del traffico urbano, AnyLogic Simulation Software

3.4.2 Diversità e realismo

La **varietà visiva** è cruciale per migliorare il realismo e l'immersione. Le principali tecniche utilizzate includono la modifica delle caratteristiche fisiche come altezza e corporatura, ogni agente viene scalato lungo gli assi X, Y e Z per creare differenze di altezza e forma fisica, *fat maps* per definire la distribuzione di grasso e muscoli sul corpo di ogni personaggio, rendendole morfologie uniche. Gli agenti possono

anche essere arricchiti con accessori come cappelli, occhiali, zaini o borse, assegnato in modo casuale o basato su regole, anche le texture e i colori degli abiti sono variati dinamicamente usando palette di colori predefinite, tecniche di *UV mapping* (processo utilizzato per proiettare una texture 2D su una superficie 3D) permettono di applicare dettagli personalizzati su ogni agente. Algoritmi procedurali generano automaticamente folle eterogenee combinando parametri visivi e comportamentali e con tecniche di randomizzazione garantiscono che nessun agente sia identico, bilanciando realismo e prestazioni.

Figura 3.5: Simulazione della diversità della folla in 3DS Max

La **Varietà comportamentale** rende le folle più realistiche attraverso la simulazione di azioni specifiche e modelli sociali, come camminare, correre e fermarsi, gli agenti seguono cicli di animazione che possono essere personalizzati per velocità, postura e andatura. Ad esempio, la velocità di corsa può essere modulata per evitare collisioni (come nel caso urbano). Gli agenti possono inoltre interagire con l'ambiente, fermandosi per osservare oggetti, interagendo con altri personaggi o esplorando percorsi alternativi. Fondamentali i modelli di interazione sociale, come Leader-Follower, in cui Un agente "leader" guida un gruppo verso un obiettivo comune, con i membri del gruppo che seguono regole di prossimità e coesione, o come la formazione di gruppi in cui gli agenti possono aggregarsi in gruppi basati su relazioni sociali (famiglie, amici) e rimanere vicini anche durante movimenti complessi. Una tematica più profonda considera l'interazione emotiva, nella quale gli agenti reagiscono ai comportamenti altrui, influenzati da emozioni come paura o calma (es. una folla che scappa in caso di pericolo). Modelli più avanzati emulano anche le forze sociali, con quindi gli agenti si muovono secondo modelli che includono forze di attrazione e repulsione, rappresentando interazioni naturali tra individui e ostacoli, e l'evitamento delle collisioni, che sfrutta tecniche come i *Reciprocal Velocity Obstacles (RVO)*, che calcolano percorsi in tempo reale per evitare scontri.

3.4.3 Modelli fondamentali

Il primo modello di base da menzionare è il **Boids Model**, creato da Craig Reynolds nel 1987, è stato originariamente sviluppato per simulare il comportamento di stormi di uccelli e banchi di pesci. I suoi principi chiave sono 3, l'allineamento (ogni individuo, o "boid", si orienta nella direzione media dei suoi vicini), coesione (ogni boid cerca di avvicinarsi al centro di massa del suo gruppo locale) e separazione (ogni boid mantiene una distanza minima dagli altri per evitare collisioni). Il modello permette a ogni boid di adattarsi dinamicamente al comportamento dei vicini senza una supervisione centrale, questo approccio decentralizzato genera movimenti complessi emergenti che sembrano realistici. È stato utilizzato in film come Batman Returns per creare sciami realistici di pipistrelli e nei videogiochi, è utile per simulare gruppi di NPC che interagiscono tra loro in modo credibile.

I secondi modelli di base sono quelli **particellari** (Helbing et al., 2000), introdotti da Dirk Helbing e colleghi, trattano ogni individuo come una particella influenzata da forze fisiche e sociali, come l'attrazione (forza che guida verso obiettivi, come un'uscita o un punto di interesse) e la repulsione (forza che evita collisioni con altri agenti o ostacoli). Sono utilizzati in simulazioni di evacuazioni in situazioni di emergenza, un esempio specifico è il modello della forza sociale (Social Force Model), che simula le dinamiche della folla in base a equazioni differenziali.

Terzo modello di base sono i **grafi di navigazione**, che rappresentano lo spazio come una rete di nodi interconnessi, nel quale ogni nodo rappresenta un punto accessibile dello spazio, gli archi tra i nodi indicano percorsi percorribili ed algoritmi come A* o Dijkstra trovano il percorso più breve tra due nodi. Utilizzati in simulazioni in ambienti complessi (es. città, edifici) e nei videogiochi per il pathfinding degli NPC.

Un primo modello più avanzato sfrutta le tecniche basate su **agenti**, nelle quali gli agenti sono entità autonome dotate di percezione, capacità decisionale (decision-making) e azioni, quindi ogni agente può percepire l'ambiente circostante (es. ostacoli, altri agenti) e prendere decisioni basate su regole locali. Degli algoritmi stocastici introducono variabilità nel comportamento per simulare decisioni non deterministiche, ampiamente utilizzati nelle simulazioni sociali come famiglie, gruppi, e gerarchie (es. leader-follower) e nella modellazione di folle eterogenee o situazioni di emergenza, dove il comportamento collettivo può emergere da interazioni locali.

L'ultimo modello avanzato più diffuso si basa invece su **flussi continui** utilizzati per simulare folle molto dense, dove il comportamento individuale è meno rilevante rispetto ai movimenti collettivi, quindi la folla viene trattata come un fluido continuo, e il movimento viene modellato con equazioni differenziali simili a quelle della dinamica dei fluidi. Largo impiego nell'analisi di flussi pedonali in stazioni ferroviarie, aeroporti o strade urbane ed in simulazioni di emergenza (es. evacuazioni di massa). Un vantaggio si quesi modelli è la loro efficienza computazionale rispetto ai modelli basati su agenti, specialmente per folle molto grandi.

Nella tabella 3.1 è possibile osservare una panoramica riassuntiva degli obiettivi e delle applicazioni dei modelli precedentemente menzionati:

Modello	**Focus**	**Applicazioni principali**
Boids Model	Allineamento, coesione, separazione	Stormi, banchi di pesci, animazioni collettive
Modelli particellari	Interazioni fisiche e sociali	Evacuazioni, movimenti pedonali
Grafi di navigazione	Pathfinding ottimale	Ambienti complessi, videogiochi
Tecniche su agenti	Autonomia e decisioni locali	Simulazioni sociali, emergenze
Flussi continui	Movimento collettivo	Folle dense, infrastrutture pedonali

Tabella 3.1: Confronto tra i modelli fondamentali nella simulazione delle folle.

3.4.4 Folle ed Intelligenza Artificiale

In questa sottosezione verranno accennati ma non approfonditi le integrazioni della simulazione delle folle con l'Intelligenza Artificiale, concentrando l'attenzione su Machine Learning (ML), Reinforced Learning (RL) ed AI Generativa.
Il **Machine Learning** è usato per analizzare grandi dataset di comportamento umano (specialmente video) per identificare modelli di movimento e generare simulazioni più realistiche, ad esempio, gli algoritmi di clustering come K-means sono usati per raggruppare comportamenti simili in scenari complessi. Altri algoritmi di ML, invece, sono impiegati per predirre i percorsi pedonali utilizzando caratteristiche globali e locali, come velocità e obiettivi intermedi, questo consente di simulare comportamenti realistici anche in ambienti affollati[11]. Per ottimizzare le simulazioni vengono anche impiegate delle tecniche di apprendimento incrementale che migliorano costantemente i modelli utilizzando nuovi dati osservati, permettendo

una simulazione adattiva e basata sui dati reali [9].

Una branca del ML, ovvero il **Reinforcement Learning** (RL), contribuisce al panorama della simulazione delle folle offrendo algoritmi che permettono agli agenti di apprendere strategie di navigazione tramite un sistema di ricompense su tentativi ed errori, ottimizzando il comportamento verso un obiettivo. Gli agenti ricevono ricompense per comportamenti desiderati (es. raggiungere una destinazione) e penalità per azioni indesiderate (es. collisioni), questo li guida verso decisioni più realistiche. Gli algoritmi RL sono usati per navigare in scenari dinamici e mutevoli, come evacuazioni di emergenza o città dense, ad esempio, il Q-learning viene adattato per gestire molteplici agenti che apprendono simultaneamente, nonostante l'ambiente in continua evoluzione. Per ridurre lo spazio degli stati e velocizzare l'apprendimento, si utilizzano tecniche come il tile coding, che suddivide gli stati in intervalli semplificati senza compromettere la qualità delle strategie apprese [12].

L'**AI Generativa**, infine, può creare variazioni visive (es. aspetto fisico, abbigliamento) e comportamentali negli agenti per aumentare la varietà e il realismo, ad esempio, i modelli generativi possono essere usati per simulare emozioni o percorsi non predefiniti, ampliando l'unicità degli agenti. L'AI Generativa utilizza modelli avanzati come le reti generative avversarie (GAN) per creare contenuti realistici e variegati [11] [12]. Questi modelli permettono agli agenti di reagire dinamicamente a situazioni specifiche, come panico o cooperazione, basandosi su dati storici o generati automaticamente.

3.4.5 Sfide e limiti attuali

Le simulazioni di folle, proporzionalmente alla complessità e livello di dettaglio, possono risultare computazionalmente dispendiose, per questo motivo sono emersi tre principali limiti:
Scalabilità: simulare folle di grandi dimensioni, che possono includere migliaia o persino milioni di agenti, richiede enormi risorse computazionali. La gestione delle interazioni individuali tra agenti diventa un problema complesso e oneroso. Ogni agente deve elaborare informazioni su percorsi, interazioni con ostacoli e altri agenti in tempo reale, causando un aumento esponenziale del costo computazionale. Una soluzione parziale comprende i livelli di dettaglio (LOD), ovvero ridurre la complessità del comportamento degli agenti più lontani dalla visuale. Anche i modelli ibridi potrebbero alleggerire il problema, combinano simulazioni basate su agenti per le interazioni locali e modelli continui (ad esempio, flussi pedonali) per descrivere le

macro-dinamiche delle folle.

Realismo comportamentale: rappresentare emozioni e decisioni complesse è un problema aperto. Molti modelli attuali si concentrano su movimenti fisici (come evitare ostacoli) piuttosto che su emozioni o motivazioni realistiche (es. paura, cooperazione, curiosità), mancano modelli psicologici accurati integrati nella simulazione e ci sono limiti nella personalizzazione, siccome molti agenti seguono comportamenti generici e non differenziati. Alcune soluzioni prevedono modelli emotivi che Integrano stati emotivi (ad esempio, panico o calma) per rendere il comportamento più credibile, ed il Machine Learning, utilizzando dati reali per allenare modelli in grado di simulare decisioni realistiche.

Prestazioni in tempo reale: la simulazione in tempo reale richiede ottimizzazioni significative per funzionare su hardware limitato, come i computer domestici o le console di gioco, gestire ambienti complessi con molteplici interazioni (es. città dense, evacuazioni di emergenza) e renderizzare graficamente le folle senza perdere il framerate. Soluzioni adottabili sono la precomputazione, ovvero il calcolo anticipato di percorsi e comportamenti statici per ridurre il carico computazionale durante la simulazione, la parallelizzazione, quindi l'uso di CPU multicore e GPU per suddividere i calcoli tra più processori, ed infine eseguire ottimizzazioni hardware-specifiche, tecniche specifiche per sfruttare al massimo le capacità delle console o dei dispositivi mobili.

Capitolo 4

Generazione e animazioni su curve

In questo capitolo verranno analizzati da un punto di vista matematico i metodi utilizzati per creare le corsie delle strade disegnate dall'utente prima di avviare la simulazione, successivamente verrà analizzato a livello teorico il procedimento necessario ad animare degli agenti in presenza di curve.

4.1 Teoria delle Curve di Bézier

Prima di analizzare i metodi utilizzati e i risultati ottenuti per creare strade e corsie, in questa sezione viene proposta la teoria di base delle curve di Beziér, ovvero il metodo utilizzato per implementare la soluzione a questo problema, grazie ad esse, infatti, è possibile disegnare curve fluide e facilmente controllabili.

4.1.1 Curve di Bézier come Polinomi

Le curve di Bézier sono definite utilizzando i *polinomi di Bernstein*, che fungono da base per rappresentare la curva. La formula generale per una curva di Bézier di grado n è:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i, \quad t \in [0,1]$$

dove:

- $\mathbf{P}_i$ sono i punti di controllo;
- $\binom{n}{i}$ è il coefficiente binomiale;
- t è il parametro che varia tra 0 e 1.

4.1.2 Curve di Bézier come Curve Parametriche

La curva di Bézier è parametrizzata in funzione di t, con ogni componente $x(t)$ e $y(t)$ rappresentata separatamente. In uno spazio bidimensionale:

$$x(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i x_i, \quad y(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i y_i$$

dove x_i e y_i sono le coordinate dei punti di controllo.
Le curve non approssimano punti specifici, bensì passano solo attraverso i punti estremi $\mathbf{P}_0$ e $\mathbf{P}_n$, senza necessariamente attraversare punti intermedi. Inoltre, si piegano verso i punti di controllo ed il poligono si controllo formato da questi approssima la forma della curva.

4.1.3 Esempio di Curva di Bézier

Di seguito è riportato un esempio esplicativo in cui consideriamo una curva di Bézier cubica ($n = 3$) con poligono di controllo:

$$\mathbf{B}(t) = (1-t)^3 \mathbf{P}_0 + 3(1-t)^2 t \mathbf{P}_1 + 3(1-t) t^2 \mathbf{P}_2 + t^3 \mathbf{P}_3$$

Supponiamo di avere i seguenti punti di controllo:

$$\mathbf{P}_0 = (0,0), \quad \mathbf{P}_1 = (1,2), \quad \mathbf{P}_2 = (2,2), \quad \mathbf{P}_3 = (3,0)$$

Il poligono di controllo è dato dalla sequenza:

$$(0,0) \to (1,2) \to (2,2) \to (3,0)$$

Di seguito in figura 4.1 è mostrato il grafico della curva Bézier cubica insieme al suo poligono di controllo:

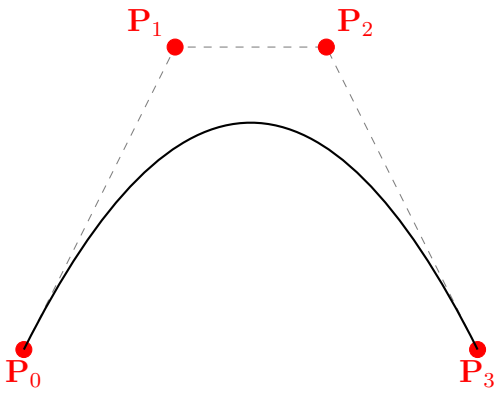


Figura 4.1: Curva di Bézier cubica con poligono di controllo

4.2 Generazione delle curve

Gli algoritmi utilizzati in questo progetto per la creazione delle corsie stradali sfruttano l'interpolazione, una tecnica che prevede il passaggio delle curve per tutti i punti di controllo che la compongono e l'approssimazione, per la quale le curve sono solamente influenzate dai punti di controllo, la figura 4.2 fornisce l'idea graficamente:

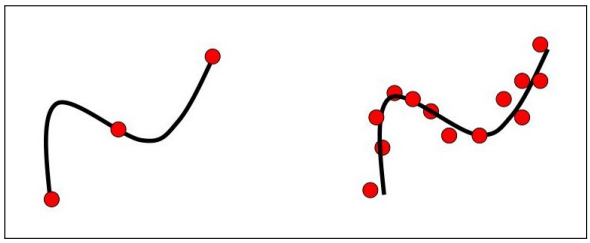


Figura 4.2: Differenza tra interpolazione (sinistra) e approssimazione (destra)

L'idea di fondo che viene utilizzata nel progetto per la creazione delle corsie stradali, partendo dalle singole curve che inserisce l'utente, è utilizzare questi due metodi appena descritti inizialmente per creare una curva di "perno", centrale, dalla quale partire a generare attraverso la copia dei punti di controllo tutte le altre corsie. Siccome l'utente fornisce manualmente le curve iniziali che collegano gli incroci, a queste basterà applicare la copia e la proiezione sia a destra che a sinistra (dipendentemente al numero di corsie destre e sinistre specificate dall'utente) dei punti di controllo della curva originale per poi applicare un'interpolazione. All'interno degli incroci, invece, c'è un problema precedente da risolvere, ovvero quello di non avere la curva originale a cui applicare l'operazione di copia ed interpolazione, quindi il modo di operare prevede inizialmente la creazione di questa curva andando a calcolare dei punti di controllo ai quali applicare un'approssimazione. Così facendo è

poi possibile agire nello stesso modo applicando alla curva appena creata le stesse operazioni di copia ed interpolazione. Queste curve "di perno" verranno chiamate curve generative, per sottolineare il loro ruolo nella creazione delle corsie.

4.2.1 Curve di Bézier di interpolazione

In figura 4.3 un primo risultato della curva generativa e dei suoi punti di interpolazione, da ribadire che è stata disegnata solo a scopo dimostrativo e di debug:

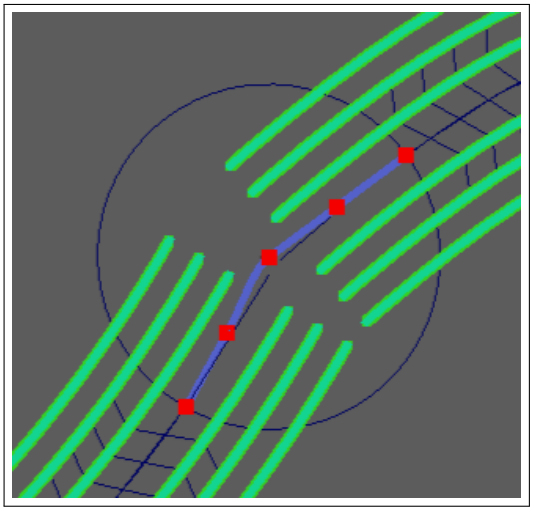


Figura 4.3: I 5 punti di interpolazione (in rosso) calcolati per creare la curva di Bézier centrale all'interno di un incrocio (in violetto)

Per poi copiarne i punti, traslarli sopra e sotto (dipendentemente dal numero di corsie per senso di marcia) ed interpolarli per ottenere il risultato in figura 4.4:

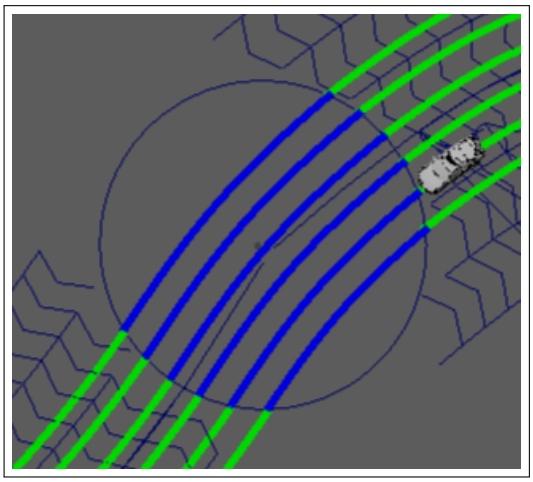


Figura 4.4: Risultato della copia e della traslazione delle curve all'interno degli incroci

4.2.2 Interpolazione TCB

La TCB-spline, che sta per Tension-Continuity-Bias spline, è una particolare variante delle Cardinal spline (o Catmull-Rom spline) che offre maggiore controllo sulla forma della curva attraverso tre parametri chiave: Tension, Continuity e Bias. Questi parametri consentono di regolare il comportamento della curva in prossimità dei punti di controllo, fornendo flessibilità nella modellazione di animazioni, superfici e traiettorie. I parametri sono appunto tre, ovvero:

- Tension (tensione): controlla la curvatura nei punti di interpolazione, valori di tensione bassi (vicini a zero) rendono la curva più fluida e "rilassata", valori di tensione alti rendono la curva più tesa, avvicinandola a una poligonale, riducendo l'arco di collegamento tra i punti di controllo. Il suo effetto è rappresentato in figura 4.5;

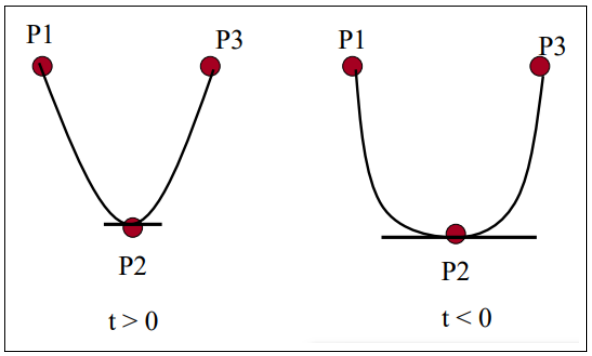


Figura 4.5: Effetto del parametro tension

$$L_i = R_i = (1-t)0.5[(P_{i+1} - P_i) + (P_i - P_{i-1})]$$

- Continuity (continuità): controlla la continuità della curca nei punti di interpolazione, valori alti di continuità danno una transizione più brusca tra i segmenti della curva, causando cambiamenti repentini di direzione, valori bassi mantengono una transizione più liscia e graduale tra i segmenti. Il suo effetto è rappresentato in figura 4.6;

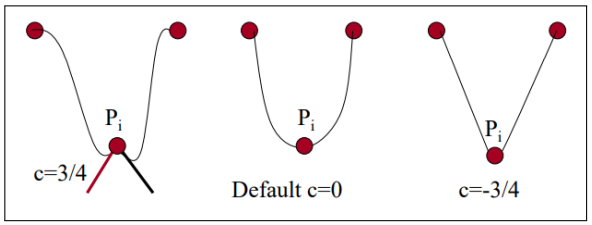


Figura 4.6: Efetto del parametro continuity

$$L_i = (1-c)0.5(P_i - P_{i-1}) + (1+c)0.5(P_{i+1} - P_i)$$

$$R_i = (1+c)0.5(P_i - P_{i-1}) + (1-c)0.5(P_{i+1} - P_i)$$

- Bias (sbilanciamento): controlla la direzione del percorso, un valore positivo di bias spinge la curva verso il punto successivo (creando una sorta di "anticipazione"), un valore negativo allontana la curva dal punto successivo, dando più enfasi al punto precedente. Il suo effetto è rappresentato in figura 4.7;

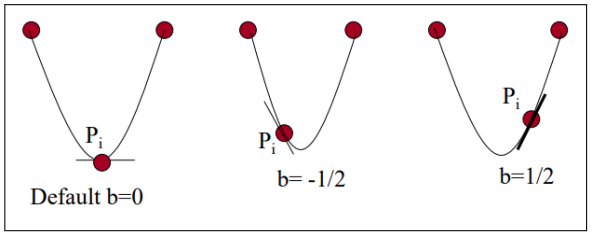


Figura 4.7: Effetto del parametro bias

$$L_i = R_i = (1+b)0.5(P_i - P_{i-1}) + (1-b)0.5(P_{i+1} - P_i)$$

La combinazione di questi tre parametri produce questi risultati:

$$L_i = \frac{(1-t)(1-c)(1+b)}{2}(P_i - P_{i-1}) + \frac{(1-t)(1+c)(1-b)}{2}(P_{i+1} - P_i)$$

$$R_i = \frac{(1-t)(1+c)(1+b)}{2}(P_i - P_{i-1}) + \frac{(1-t)(1-c)(1-b)}{2}(P_{i+1} - P_i)$$

Una volta calcolati i valori per L_i e R_i, la costruzione per i punti di controllo interni della curva di Bézier sono i seguenti:

$$p_i^+ = P_i + \frac{1}{3}R_i \quad \text{e} \quad p_i^- = P_i - \frac{1}{3}L_i$$

4.2.3 Pseudo-codice della funzione che implementa le TCB-Spline `interpolate_curves`

La funzione `interpolate_curves`, calcola i punti intermedi su una curva utilizzando la tecnica delle TCB-spline (Tension-Continuity-Bias). Restituisce un vettore di punti interpolati che verranno utilizzati dalla funzione `createParallelLanes` (analizzato nel prossimo capitolo) per creare le corsie di ogni strada, sia dentro che fuori agli incroci.

Passaggi principali

1. **Inizializzazione**

   - Crea un vettore vuoto `curvePoints` per memorizzare i punti della curva interpolata.
   - Determina il numero di punti di controllo `n`:

   $$n = \text{points.size()}$$

   - Se $n < 2$, ritorna un vettore vuoto (non ci sono abbastanza punti per una curva).

2. **Calcolo delle tangenti per ciascun punto di controllo**

   - Per ogni punto di controllo P_i (con $i = 1$ a $n - 2$):
     (a) Ottiene i punti P_{i-1} (precedente) e P_{i+1} (successivo).
     (b) Calcola le tangenti sinistra (L_i) e destra (R_i) usando:

   $$L_i = (P_i - P_{i-1}) \cdot \frac{(1 - \text{tension})(1 - \text{continuity})(1 + \text{bias})}{2} + (P_{i+1} - P_i) \cdot \frac{(1 - \text{tension})(1 + \text{continuity})(1 - \text{bias})}{2},$$

   $$R_i = (P_i - P_{i-1}) \cdot \frac{(1 - \text{tension})(1 + \text{continuity})(1 + \text{bias})}{2} + (P_{i+1} - P_i) \cdot \frac{(1 - \text{tension})(1 - \text{continuity})(1 - \text{bias})}{2}.$$

3. **Calcolo dei punti intermedi**

   - Calcola i punti intermedi:

   $$P_i^- = P_i - \frac{L_i}{3}, \quad P_i^+ = P_i + \frac{R_i}{3}$$

   - Aggiunge i punti P_i^-, P_i, e P_i^+ al vettore `curvePoints`.

4. **Risultati**

   - Restituisce il vettore `curvePoints` contenente i punti interpolati, che include sia i punti originali che quelli intermedi calcolai (P_i^- e P_i^+).

4.3 Animazione di oggetti lungo traiettorie rappresentate da curve

L'animazione sulle curve è una tecnica utilizzata in computer graphics per far muovere oggetti o personaggi lungo traiettorie predefinite, rappresentate da curve matematiche ed è largamente impiegata in settori come l'animazione 3D, la simulazione di fisica, i videogiochi e il cinema, in cui il controllo preciso e fluido del movimento è essenziale. Il processo di animazione è composta da alcuni passaggi fondamentali:

- Definizione del Percorso: la curva che rappresenta il percorso di un oggetto viene definita utilizzando punti di controllo o parametri specifici, a seconda del tipo di curva. L'animatore specifica i punti chiave lungo il percorso, attraverso i quali l'oggetto deve passare.

- Parametrizzazione del Movimento: il movimento lungo la curva è generalmente parametrizzato in funzione del tempo. Il parametro può essere usato per controllare la posizione dell'oggetto sulla curva in ogni istante, consentendo animazioni fluide e continue, è possibile utilizzare interpolazioni uniformi o non uniformi per ottenere variazioni nella velocità dell'oggetto mentre si muove lungo la curva (interpolazione temporale). Ad esempio, si può avere un'accelerazione o decelerazione applicando interpolazioni non lineari.

- Controllo della Velocità: la velocità lungo la curva può essere regolata attraverso la parametrizzazione del tempo. Usando una parametrizzazione uniforme, l'oggetto si muove a velocità costante. Parametrizzazioni non uniformi consentono di aumentare o diminuire la velocità a seconda del segmento della curva, per simulare accelerazioni, frenate o movimenti più complessi.

- Orientamento dell'Oggetto (Tangenti e Frenet Frame): quando un oggetto si muove lungo una curva, il suo orientamento può essere controllato utilizzando le tangenti alla curva. Le tangenti forniscono la direzione del movimento in un dato punto, mentre un Frenet Frame può essere usato per calcolare l'orientamento dell'oggetto nello spazio 3D, tenendo conto della curvatura della traiettoria.

Capitolo 5

Gestione delle strade

Prima di entrare nel dettaglio della logica della costruzione delle strade, è da specificare la modalità di visualizzazione 3D delle corsie provvisoria utilizzata, ovvero viene invocata una funzione di nome `draw` all'interno della classe `TrafficModule` che permette di tracciare segmenti, disegnare punti, dati e contenuto testuale partendo dagli oggetti messi a disposizione da Atoms. Questa funzione è stata utilizzata principalmente a scopo illustrativo e di debug per risaltare i risultati della creazione delle corsie e degli agenti.

5.1 Creazione corsie multiple e algoritmi

Il processo di costruzione delle strade e delle corsie comincia dalle curve e dai dati che l'utente inserisce durante la fase di creazione e modifica del contesto urbano accennato precedentemente. Infatti, dopo aver collegato strade e incroci, specificato la larghezza ed il numero delle corsie per senso di marcia è possibile utilizzare le curve generative nel seguente modo:

- in Atoms gli oggetti che rappresentano le curve sono identificati come un vettore di punti nello spazio tridimensionale, l'idea è quella di copiare e traslare lungo la normale, mantenendo come distanza la larghezza della corsia inserita dall'utente, un certo numero di punti presi dalla curva generatrice per poi usarli per costruire quelle che sono le nuove corsie;
- una volta creato il vettore di punti, questo viene ispezionato per eliminare eventuali intersezioni con se stesso che porterebbero alla formazione di "riccioli" nel momento del disegno della curva finale;
- il vettore risultante viene poi utilizzato per creare la curva TCB finale che verrà infine disegnata.

Di seguito una breve spiegazione delle funzioni create per ottenere i risultati riportati graficamente in figura 5.1.

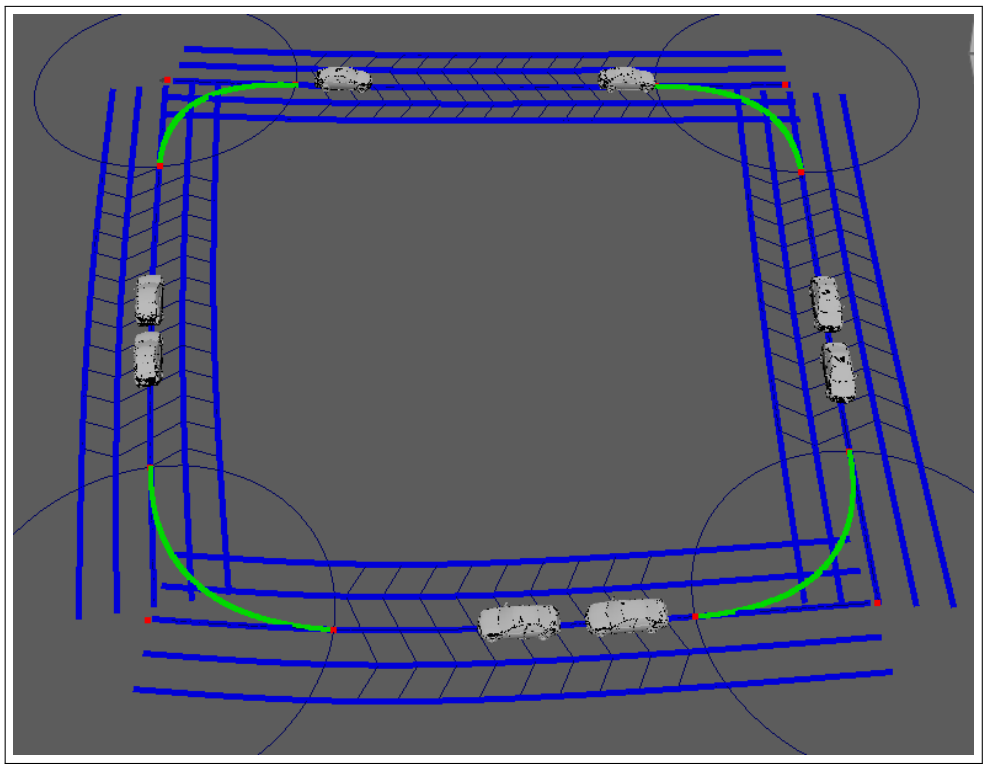
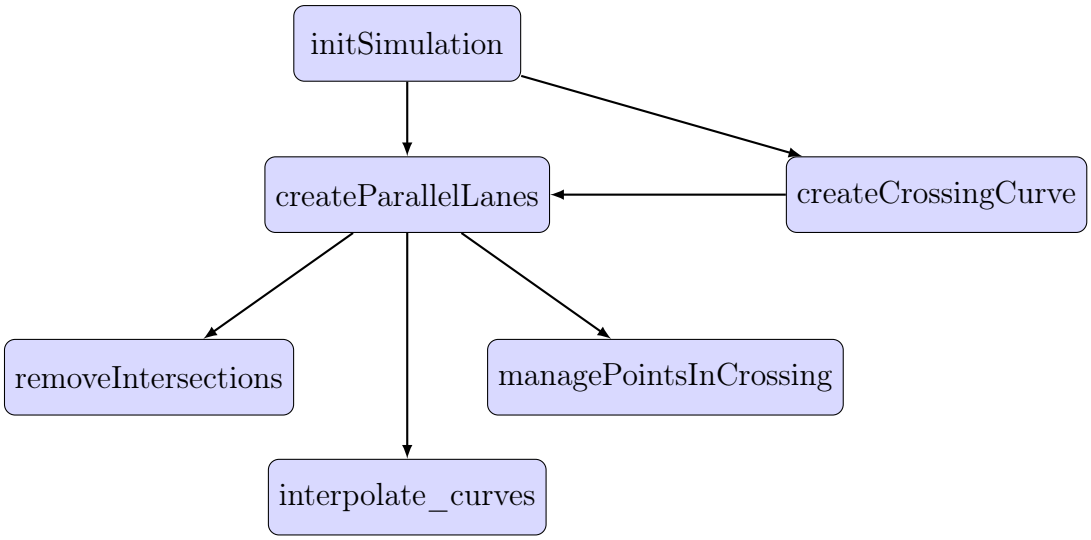


Figura 5.1: Semplice creazione delle corsie delle strade (in blu) e del primo collegamento tra esse all'interno degli incroci (in verde)

Per creare corsie blu della figura 5.1, è stata invocata la funzione `createParallelLanes` passando come parametri di input le quattro strade (originariamente semplici curve di Maya) disegnate manualmente e le relative informazioni come larghezza e numero delle corsie ecc.; di questa funzione verrà fornito lo pseudo-codice nelle sezioni successive del capitolo.
Per le curve verdi invece, ovvero le curve generatrici all'interno degli incroci, è stato utilizzato il metodo `createCrossingCurve`, di cui verrà anch'esso spiegato lo pseudo-codice, ma prima ecco riportato nel seguente schema a blocchi il flusso di esecuzione dei metodi che generano le corsie dello scenario urbano:



La prima ad essere invocata è la funzione `initSimulation`, la quale chiama `createParallelLanes` per generare le corsie delle strade disegnate dall'utente, che a sua volta utilizza `removeIntersections`, `managePointsInCrossing` e `interpolate_curves`, subito dopo vengono create le curve generatrici all'interno degli incroci con `createCrossingCurve`, la quale sfrutta anch'essa successivamente `createParallelLanes` per generare le corsie anche all'interno degli incroci. Ecco una breve spiegazione del contenuto dei blocchi, che corrispondono ciascuno ad una delle seguenti funzioni:

- `initSimulation`: funzione principale che viene invocata nel momento in cui vengono apportate delle modifiche ai progetto Maya in uso, è responsabile della creazione di tutti i moduli inerenti a generazione di corsie e agenti, il suo pseudo-codice verrà discusso nel dettaglio nelle seguenti sezioni;

- `createParallelLanes`: metodo che copia i punti della strada originale creata dall'utente e li trasla spostandoli alla distanza della larghezza delle corsie, dato anch'esso preso in input dall'utente. Il risultato è un vettore di curve la cui grandezza è la somma del numero delle corsie per senso di marcia della strada che viene passata come parametro;

- `createCrossingCurve`: metodo che date due curve generative disegnate dall'utente forma la curva generativa che le collega all'interno degli incroci;

- `removeIntersections`: metodo utilizzato nel momento in cui viene creata una curva, per cancellare ogni punto che vada ad intersecare la retta stessa in caso fossero presenti curve particolarmente angolate, l'effetto è il seguente riportato in figura 5.2:

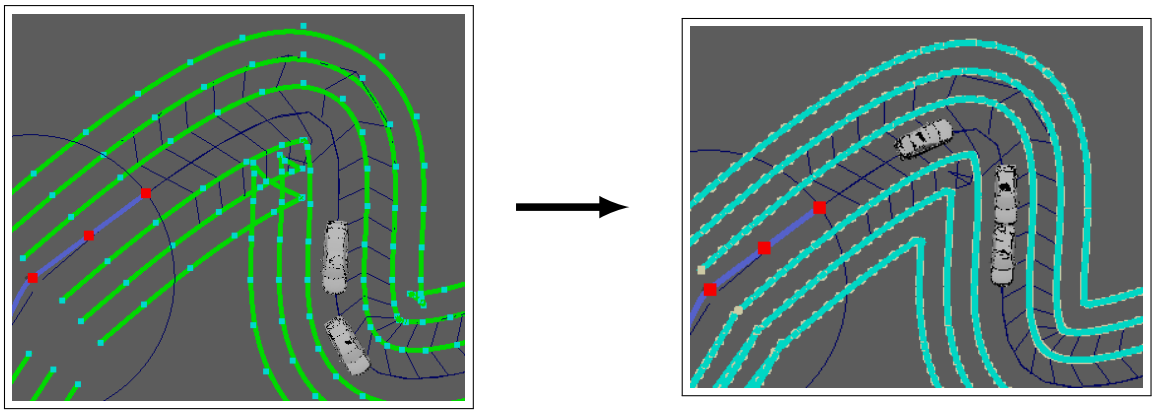


Figura 5.2: Risultato del metodo `removeIntersections`

- `interpolate_curves`: dato un vettore di punti, utilizza il metodo TCB (tensione, continuity, bias) per interpolarli e creare un vettore dal quale si può creare la curva parallela richiesta, lo pseudo-codice di questa funzione è stato analizzato nel capitolo precedente;

- `managePointsInCrossings`: funzione che in caso di corsie esterne agli incroci elimina tutti i punti che si trovano all'interno di questi, mentre in caso di corsie interne agli incroci elimina tutti i punti esterni a questi, il risultato riportato in figura 5.3, nella quale si può osservare che le curve blu non escono dalla circonferenza dell'incrocio, mentre quelle verdi non entrano:

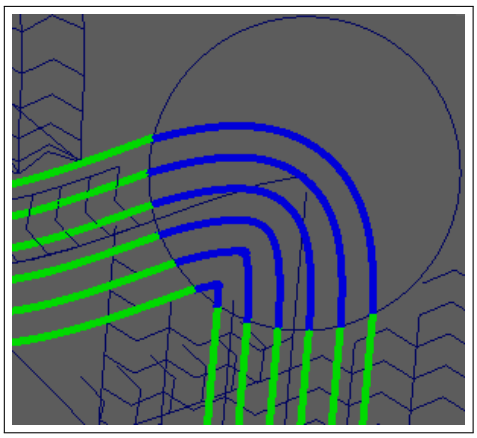


Figura 5.3: Effetto della funzione `managePointsInCrossing`

5.1.1 Pseudo-codice della funzione principale `initSimulation`

La prima funzione che viene chiamata non appena i parametri dell'interfaccia di Atoms vengono modificati, è `initSimulation`, che ha lo scopo di inizializzare una

simulazione di traffico, configurando corsie e curve per strade e incroci, e generando le strutture dati necessarie.

Passaggi principali

1. **Recupera i parametri dal metadata**:
   - Estrae `trafficName`, `agentType`, `iterations`, `maxNumAgents`, `distance`, `linearNoise`.
2. **Inizializza il generatore di layout del traffico**:
   - Crea un'istanza di `TrafficLayoutGenerator` usando un seme casuale.
   - Configura il generatore con i parametri estratti.
3. **Recupera dati dal traffico**:
   - Ottiene l'oggetto `traffic` e verifica che esista.
   - Estrarre informazioni come lato guida (`side`), strade (`streets`) e incroci (`crossings`).
   - Configura il lato guida (`m_side`) in base al valore di `side`.
4. **Genera corsie per le strade**:
   - Per ogni strada:
     (a) Calcola la larghezza della corsia.
     (b) Ottiene la curva della strada, il numero di corsie e la velocità massima.
     (c) Usa `createParallelLanes` per creare corsie parallele.
     (d) Aggiunge le corsie generate ai vettori globali.
5. **Gestisci i punti all'interno degli incroci**:
   - Per ogni incrocio:
     (a) Rimuove punti stradali all'interno del raggio dell'incrocio.
     (b) Rimuove punti delle corsie di incrocio fuori dal raggio.
6. **Genera curve di connessione per gli incroci**:
   - Per ogni combinazione di strade appartenenti a un gruppo di incrocio:
     (a) Ottiene le curve delle strade da connettere.

(b) Usa `createCrossingCurve` per generare una curva di connessione.

(c) Genera corsie parallele sulla curva con `createParallelLanes`.

(d) Rimuove punti fuori dall'incrocio.

(e) Aggiorna i vettori globali con le corsie generate.

7. **Unifica le corsie**:

   - Combina corsie di strade e incroci in un unico vettore globale.

8. **Inizializza il gestore degli incroci**:

   - Crea un `CrossingsManager` usando incroci, corsie di incrocio e corsie di strada, questo oggetto si occuperà di contenere le logiche inerenti agli incroci.

9. **Configura il generatore di layout**:

   - Imposta le corsie nel generatore di layout.
   - Aggiunge il generatore al gruppo di agenti.

10. **Calcola le lunghezze delle curve**:

    - Per ogni corsia:

      (a) Calcola la lunghezza della curva e aggiungila al vettore delle lunghezze.

    - Calcola la somma totale delle lunghezze.
    - Ordina corsie e lunghezze in ordine decrescente (opzionale).

5.1.2 Pseudo-codice della funzione di creazione delle corsie per ogni strada

`createParallelLanes`

La seconda funzione fondamentale di cui è necessario fornire l'implementazione è `createParallelLanes`, che genera corsie parallele a partire da una curva di base, con offset calcolati per ogni corsia, e rimuove eventuali intersezioni, restituendo un vettore di oggetti `TrafficLane` che sarà costituito da tutte le corsie per la strada fornita in input.

Passaggi principali

1. **Inizializzazione**
   - Crea un vettore vuoto `result` per contenere le corsie generate.
   - Calcola la lunghezza della curva data (`length`).
   - Determina il numero di punti da campionare (`numPoints`), che è il risultato della divisione della lunghezza di ogni corsia per una certa costante, in questo modo curve più lunghe saranno composte da più punti.
2. **Campionamento dei punti sulla curva**
   - Calcola il passo di campionamento (`sampleStep`).
   - Per ogni valore di t da 0 a `numPoints`:
     - Calcola il punto sulla curva usando `pointOnCurve`.
     - Aggiunge il punto campionato al vettore `sampledPoints`.
3. **Generazione delle corsie parallele**
   - Per ogni corsia da generare (in base a `leftLanesNum` e `rightLanesNum`):
     (a) Inizializza i vettori `parallelCurveAbove` e `parallelCurveBelow`.
     (b) Per ogni punto campionato:
       - Calcola la tangente e la direzione perpendicolare al punto.
       - Genera il punto sopra e sotto il punto campionato usando l'offset.
       - Aggiunge i punti generati ai rispettivi vettori.
     (c) Rimuove le intersezioni dai vettori di punti usando `removeIntersections`.
4. **Interpolazione delle curve e creazione delle corsie**
   - Per ogni corsia (sinistra e destra):
     (a) Interpola i punti per creare una curva liscia (`interpolate_curves`).
     (b) Crea un oggetto `TrafficLane` con:
       - ID della corsia (`laneId`).
       - La curva interpolata.
       - Parametri come lato (`laneSide`), lunghezza, e limite di velocità.
     (c) Aggiunge l'oggetto al vettore `result`.
5. **Risultati**
   - Restituisce il vettore `result` contenente tutte le corsie generate.

5.2 Collegamenti all'interno degli incroci e pseudo-codice

Come accennato nelle sezioni e nei capitoli precedenti, la funzione `createCrossingCurve`, date due curve stradali entranti in uno stesso incrocio, genera i punti che formano il collegamento interno ad esso che collega le due strade, utilizzando altre due funzioni ausiliarie di nome `circleSegmentIntersection` e `findIntersection`, utili ad individuare i punti di intersezione delle due curve con l'incrocio stesso. Nello specifico, date due curve, la funzione crea un vettore di punti composto dai due punti di inizio e fine del segmento della prima curva che interseca con il l'incrocio, due punti in cui le curve si intersecano (altrimenti gli estremi più vicini) ed infine i due punti di inizio e fine del segmento della seconda curva che interseca l'incrocio. Fondamentale quindi, siccome al di fuori degli incroci le curve generatrici vengono create direttamente dall'utente, mentre all'interno degli incroci mancherebbero. In figura 5.4 sono riportati tre passaggi di miglioramento delle curve all'interno degli incroci.

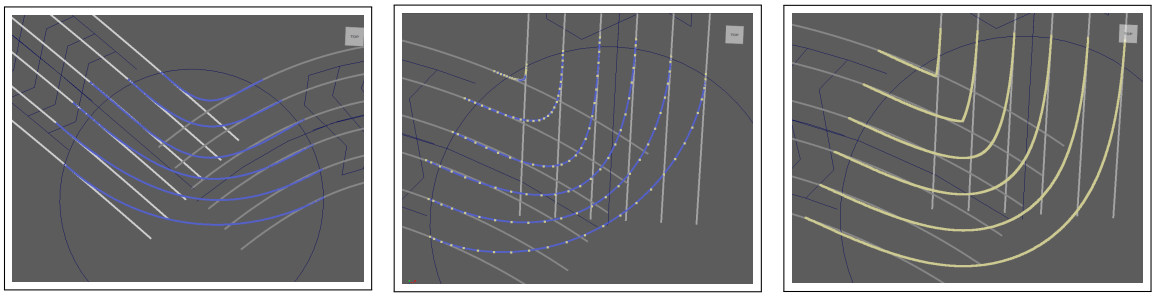


Figura 5.4: Processo di creazione e miglioramento dei collegamenti fra corsie all'interno degli incroci

Pseudo-codice dei passaggi principali della funzione `createCrossingCurve`

1. **Inizializzazione**
    - Crea un vettore vuoto `cvs` per contenere i punti di controllo della curva.
    - Inizializza due flag (`firstOk`, `secondOk`) a `false`, per indicare se le intersezioni con le curve sono state trovate.
    - Imposta due vettori (`firstIntersection`, `secondIntersection`) per memorizzare i punti di intersezione.
    - Calcola il numero di segmenti (`segmentsNum`) e i passi di campionamento (`firstStep`, `secondStep`) per le due curve.

2. **Trova le intersezioni tra le curve e il cerchio**

   - Itera attraverso i segmenti delle due curve:
     - Per la prima curva:
       (a) Verifica l'intersezione con il cerchio usando `circleSegmentIntersection`.
       (b) Se l'intersezione è trovata, memorizza i punti di inizio e fine del segmento.
     - Ripete lo stesso per la seconda curva.

3. **Verifica se entrambe le intersezioni sono state trovate**

   - Se entrambe le intersezioni sono valide:
     (a) Determina l'ordine dei punti di intersezione in base alla distanza dal centro del cerchio.
     (b) Aggiunge i punti della prima curva al vettore `cvs`.
     (c) Trova i punti di intersezione tra le due curve usando `findIntersection` e li aggiunge a `cvs`.
     (d) Aggiunge i punti della seconda curva a `cvs`, ordinandoli correttamente.

4. **Crea la curva risultante**

   - Crea un oggetto `AtomsUtils::Curve` usando il vettore `cvs`, composta dai punti di intersezione e di connessione tra le due curve iniziali, limitata al cerchio dell'incrocio specificato e restituiscilo.

5.2.1 Collegamenti multipli all'interno degli incroci

Questa sezione ha lo scopo di porre l'attenzione sulla corretta gestione del codice utilizzato precedentemente permette di ottenere anche la gestione di più strade che entrano in un solo incrocio. Il risultato finale è il seguente riportato in figura 5.5, che rappresenta lo scenario ideale di esempio che verrà utilizzato anche nei capitoli successivi della tesi:

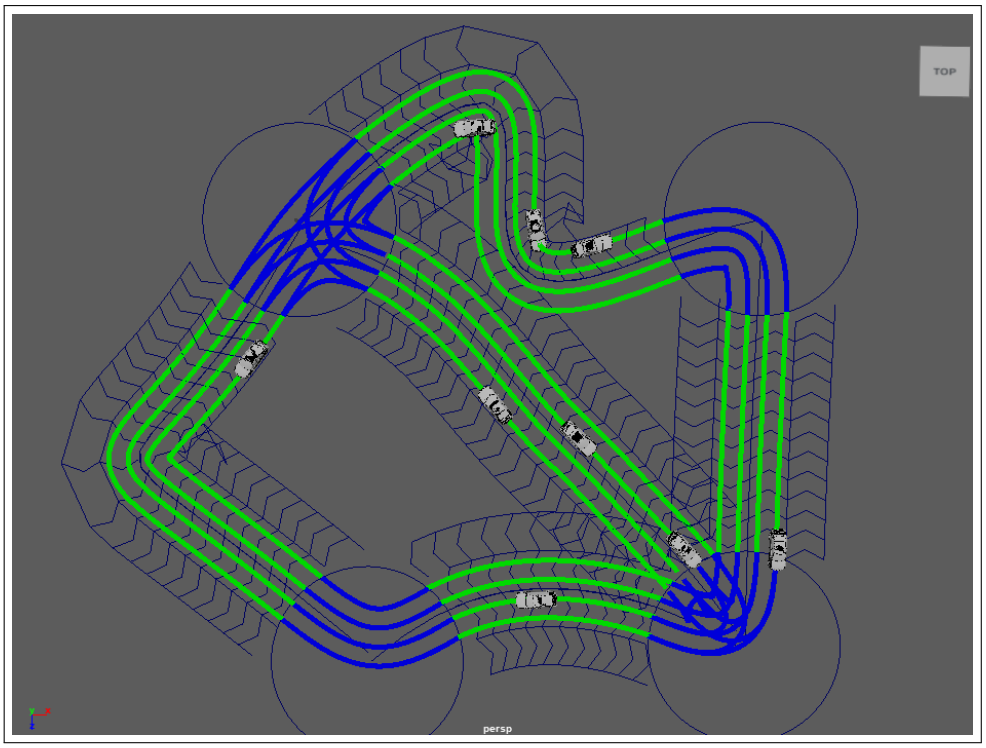


Figura 5.5: Scenario con collegamenti stradali multipli all'interno degli incroci

Capitolo 6

Generazione degli agenti

Un'agente, in generale, è un'entità autonoma e individuale in grado di percepire l'ambiente circostante, prendere decisioni basate su regole o algoritmi, e agire per raggiungere determinati obiettivi. Nella simulazione delle folle, gli agenti sono le unità fondamentali per creare comportamenti complessi e realistici ed gnuno di essi contribuisce a modellare le dinamiche globali della folla grazie alla combinazione di movimenti individuali e interazioni di gruppo. In questo progetto le folle sono identificate dai veicoli che si muovono lungo traiettorie (ovvero strade, rappresentate da curve) e la strategia di animazione è quella degli agenti, che rappresentano ognuno un'automobile.
Ogni agente simula un individuo con caratteristiche e obiettivi unici, ma prima di affrontare i loro comportamenti, analizziamo le modalità di creazione in Atoms.

6.1 Costruzione degli agenti

Come accennato nei capitoli precedente, i metadati degli agenti vengono assegnati con i valori dei diversi parametri inseriti dall'utente in fase di inizializzazione dello scenario stradale. Nello specifico i parametri personalizzabili e le loro funzioni sono:

- range di velocità minima e massima, un valore casuale calcolato in questo intervallo di valori determina la velocità con la quale l'agente comincerà la simulazione;
- range velocità di inversione minima e massima, un valore casuale calcolato in questo intervallo determina la velocità alla quale si dovrà portare l'agente in caso dovesse fare un'inversione;
- ignoranento del limite di velocità, valore booleano che determina se è possibile andare oltre i limiti di velocità previsti dalle strade che percorrono gli agenti;

- range di distanza di sicurezza minima e massima, un valore casuale calcolato in questo intervallo determina la distanza di sicurezza che devono mantenere gli agenti l'uno dall'altro;
- range di accelerazione minima e massima, un valore casuale calcolato in questo intervallo determina l'accelerazione (e decelerazione) per frame che un agente può avere;
- modalità di navigazione, seleziona l'obiettivo destinazione di ciascun agente, tra cui: punto di destinazione casuale, punto di destinazione indicato, sequenza di punti di destinazione, metadata (per es. un altro agente);
- range di sfasamento minimo e massimo, un valore casuale calcolato in questo intervallo determina l'offset sulla normale delle curve in cui viene generato l'agente, per fare in modo che il veicolo non sia proprio al centro della lane;
- massimo numero di agenti, valore che inserisce un limite superiore al numero di utenti massimo che può essere generato nella corrente simulazione, notare che non è un numero garantito;
- numero di iterazioni per la generazione degli agenti, notare che un numero più alto aumenta le probabilità che il numero massimo di agenti generabili venga raggiunto;
- range di distanza minima e massima, un valore casuale calcolato in questo intervallo determina la distanza minima in cui deve essere generato un agente, in caso contrario l'agente viene scartato.

Dopo aver inserito dall'interfaccia di Atoms i valori ai due parametri volti alla creazione degli agenti, ovvero il numero massimo di agenti generabili ed il numero massimo di iterazioni, un ciclo annidato ed un contatore genera gli agenti inserendoli nel contesto urbano creato dall'utente, esclusi gli spazi all'interno degli incroci. Il numero massimo di iterazioni è utile in caso un agente non possa essere creato per delle ragioni come per esempio la violazione della distanza minima iniziale da mantenere fra un agente e un altro. Le funzioni adibite alla creazione ed inizializzazione degli agenti sono principalmente due::

- `generate`: metodo contenuto nella classe `TrafficGenerator`, la istanza viene creata nella classe `TrafficModule` si occupa di creare tutti gli agenti basandosi sul numero massimo di agenti e di iterazioni inseriti dall'utente tramite l'interfaccia di Atoms. Ogni agente è rappresentato con un oggetto di Atoms

chiamato `AgentData`, che nel momento della creazione viene inizializzato inserendo la propria posizione nel piano, numeri identificativo univoco, tipo di agente (definisce il modello 3D della macchina), direzione e corsia corrente;

- `agentsCreated`: metodo interno alla classe `TrafficModule` che viene chiamato ogni qual volta un agente viene creato, si occupa di creare ed aggiungere tutti i metadati aggiuntivi necessari, come velocità, accelerazione, distanza di sicurezza, destinazione, percorso di corsie da seguire per raggiungere la destinazione, percorso per raggiungere la destinazione ed un valore booleano che indica se la destinazione è stata raggiunta o meno;

6.2 Selezione della posizione di spawn

La selezione della posizione di spawn di ciascun agente è delegata ad una funzione di nome `getRandomPosition`, la quale implementa un campionamento uniforme su un segmento unificato formato dall'unione di tutte le corsie, al di fuori degli incroci, che formano lo scenario urbano creato dall'utente, garantendo una distribuzione uniforme lungo la lunghezza totale delle corsie. L'idea è collegare tutte le curve che formano le corsie delle strade per creare un'unica curva la cui lunghezza è nota, siccome è semplicemente la somma delle lunghezze di tutti i segmenti che la formano. Dopodichè viene generato un numero casuale nell'intervallo La funzione `getRandomPosition` calcola una posizione casuale sulla rete stradale rappresentata da segmenti di curve, garantendo una distribuzione uniforme lungo la lunghezza totale delle corsie. Di seguito viene fornito lo pseudo-codice della funzione:

Parametri della funzione:

- `randomSeed`: una sorgente di numeri casuali utilizzata per generare un valore casuale distribuito uniformemente;
- `lengthSum`: la lunghezza totale della rete stradale, calcolata sommando le lunghezze di tutte le corsie;
- `laneCurvesLength`: ln vettore contenente le lunghezze delle singole corsie;
- `streetLanes`: ln vettore di oggetti `TrafficLane` che rappresentano le corsie.

Passaggi principali:

1. Generazione di un punto casuale : un valore casuale x viene generato nell'intervallo $[0, \text{lengthSum}]$:

$$x = \text{random}(0, \text{lengthSum})$$

2. Identificazione della corsia : per identificare la corsia corrispondente al punto casuale x, viene utilizzata una somma cumulativa:

$$\text{acc} = \sum_{i=0}^{n} \text{laneCurvesLength}[i]$$

Si trova l'indice della corsia (`laneIndex`) tale che:

$$\text{acc} \geq x$$

3. Calcolo della posizione relativa nella corsia : una volta trovata la corsia corretta, si calcola la posizione locale lungo la corsia come:

$$t = \frac{x - (\text{acc} - \text{laneCurvesLength[laneIndex]})}{\text{laneCurvesLength[laneIndex]}} \cdot \text{maxU}$$

Dove maxU è il valore massimo del parametro della curva per quella corsia.

4. Determinazione della posizione 3D : utilizzando il parametro t, la posizione 3D viene calcolata con:

$$\text{position} = \text{streetLanes[laneIndex].getCurve().pointOnCurve}(t)$$

Inoltre:

- La **direzione** lungo la corsia è data dalla tangente normalizzata:

$$\text{direction} = \frac{\text{streetLanes[laneIndex].getCurve().tangentOnCurve}(t)}{\| \cdot \|}$$

- La **direzione perpendicolare** viene calcolata nel piano orizzontale come:

$$\text{perpDirection} = (-\text{direction}.z, 0, \text{direction}.x)$$

Risultati: la funzione restituisce un oggetto `RandomPositionResult` contenente:

- La posizione 3D calcolata;

- La direzione lungo la corsia;
- La direzione perpendicolare;
- L'ID della corsia.

Dopo aver creato questo risultato, un semplice ciclo controlla che la posizione creata non abbia alcun altro agente nel raggio minimo specificato dall'utente, quindi l'agente viene creato ed aggiunto alla struttura che mantiene le informazioni di tutti i veicoli, altrimenti viene scartato e si procede con la prossima iterazione.

6.3 Pseudo-codice della funzione `agentsCreated`

La funzione configura gli agenti appena creati, assegnando loro proprietà come velocità, accelerazione e destinazioni casuali, e calcola il percorso verso la destinazione.

Passaggi principali

1. **Parallelizzazione del processo sugli agenti**
   - Esegue un ciclo parallelo su tutti gli agenti:
     (a) Ottiene il **network** dell'agente corrente.
     (b) Recupera i nodi `pelvisOperator` e `bindPoseOperator`.
     (c) Se uno dei due nodi esiste già, continua al prossimo agente.
     (d) Crea un nuovo nodo `bindPoseOperator` e assegnalo all'agente.
2. **Imposta i parametri degli agenti**
   - Per ogni agente:
     (a) **Velocità**:
       – Ottiene l'intervallo di velocità dal metadata.
       – Genera una velocità casuale e la aggiunge al metadata dell'agente.
     (b) **Accelerazione**:
       – Ottiene l'intervallo di accelerazione dal metadata.
       – Genera un'accelerazione casuale e aggiungila al metadata dell'agente.
     (c) **Distanza di sicurezza**:
       – Ottiene l'intervallo di distanza di sicurezza dal metadata.

- Genera una distanza casuale e la aggiungi al metadata dell'agente.

(d) **Destinazione casuale**:
- Usa `getRandomPosition` per generare una destinazione casuale sulla rete stradale.
- Aggiunge la destinazione al metadata dell'agente.
- Salva la posizione della destinazione in `m_destinationPoints`.

(e) **Percorso verso la destinazione**:
- Usa l'algoritmo A* per calcolare il percorso dal punto iniziale alla destinazione.
- Aggiunge il percorso al metadata dell'agente.

(f) Imposta la proprietà `destinationReached` su `false`.

Risultati

Ogni agente ha:

- Velocità, accelerazione e distanza di sicurezza impostate casualmente.
- Una destinazione casuale sulla rete stradale.
- Un percorso calcolato verso la destinazione.
- La proprietà `destinationReached` inizialmente impostata su `false`.

Di seguito in figura 6.1 un esempio di scenario creato utilizzando un elevato numero di agenti:

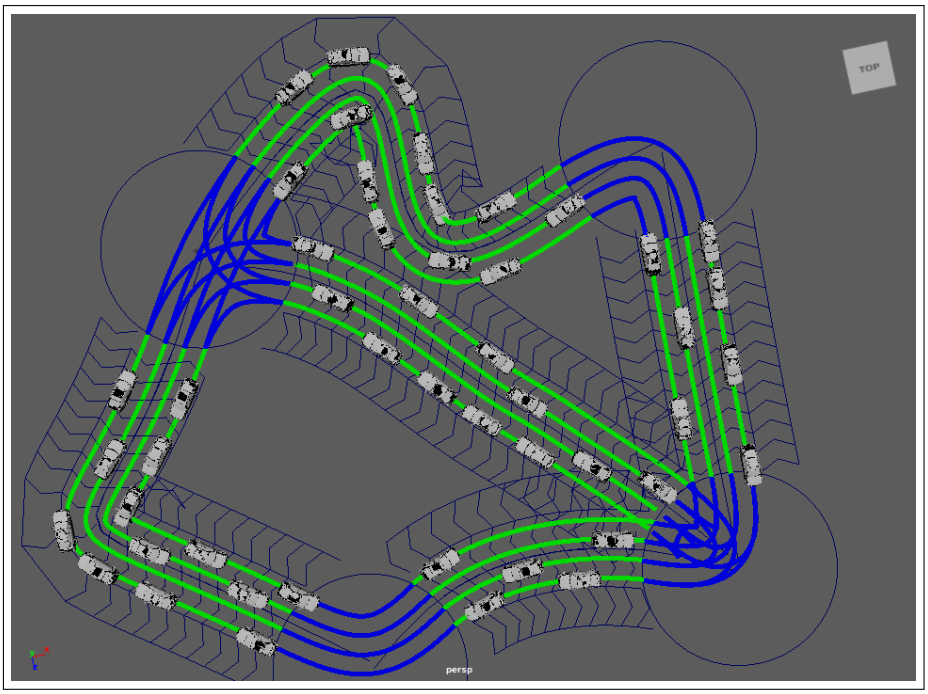


Figura 6.1: Scenario con un numero elevato di agenti

Capitolo 7

Comportamento degli agenti

Come anticipato nel capitolo precedente, gli agenti sono entità con caratteristiche e obiettivi unici, come per esempio muoversi da un punto A ad un punto B evitando ostacoli, cambiare la propria velocità a seconda di alcuni fattori come, in contesto urbano, rallentare per evitare collisioni o accelerare per effettuare un sorpasso. Possono anche agire come parte di un gruppo, come seguire un leader o un obiettivo comune, attuare comportamenti cooperativi, comunicativi o conflittuali, oppure, nel contesto urbano, formando una coda ordinata ad un semaforo.

7.1 Raggiungimento di una destinazione (algoritmo A*)

Ad ogni frame durante la simulazione, viene invocato il metodo `initFrame` all'interno della classe `TrafficModule` che si occupa di aggiornare la posizione di ogni agente, muovendoli sulla base di direzione e velocità frame dopo frame. La funzione sfrutta l'esecuzione parallela per poter gestire efficientemente un numero elevato di agenti. Ogni agente conterrà come metadata un vettore contenente tutte le corsie che dovrà percorrere per raggiungere la propria destinazione, il quale viene calcolato già all'interno del metodo `agentCreated`, nel quale verrà applicato l'algoritmo A* alla mappa delle corsie dello scenario corrente.
Algoritmo che dati in input l'ID della corsia di partenza, la posizione nello spazio di destinazione e l'ID della corsia di destinazione, crea un vettore di ID di corsie che rappresenta il percorso più breve per raggiungere la destinazione. Dettagli dell'algoritmo:

L'algoritmo A* calcola il percorso ottimale in una rete stradale rappresentata come un grafo di corsie connesse. La sua implementazione si basa sui seguenti passaggi e componenti:

Fasi principali

1. **Inizializzazione**:
   - Crea un dizionario `gScore` per memorizzare il costo più basso conosciuto per raggiungere ogni corsia.
   - Tutte le corsie hanno un costo iniziale pari a infinito, eccetto la corsia di partenza che ha un costo pari a 0.
2. **Coda prioritaria**:
   - Utilizza una `priority_queue` per ordinare le corsie in base al loro costo totale (`totalCost`).
   - La corsia di partenza viene inserita nella coda con un costo totale iniziale calcolato tramite l'euristica (distanza dal centroide della corsia di partenza alla destinazione).
3. **Iterazione principale**:
   - Finché la coda prioritaria non è vuota:
     (a) Viene estratto il nodo con il costo totale minore.
     (b) Se il nodo corrente è la corsia di destinazione (`endLaneId`), viene ricostruito e restituito il percorso ottimale.
     (c) Altrimenti, vengono esplorati i vicini della corsia corrente.
4. **Esplorazione dei vicini**:
   - Ogni corsia adiacente viene valutata tramite `TrafficLaneLink`.
   - Viene calcolato il nuovo costo per raggiungere ogni corsia vicina.
   - Se il nuovo costo è inferiore a quello già noto, aggiorna il percorso migliore e inserisce la corsia nella coda prioritaria.

Componenti implementative

1. **Vettore di `TrafficLaneLink`**:
   La realizzazione dell'algoritmo ha richiesto la creazione di questa struttura in

grado di rappresentare la topologia del contesto stradale utilizzato. Oggetto contenente 4 campi:

- Corsia entrante nell'incrocio.
- Corsia uscente dall'incrocio.
- Corsia interna di collegamento.
- ID dell'incrocio.

2. **Funzione** `heuristic`:
   - Calcola una stima del costo per raggiungere la destinazione finale.
   - Utilizza la distanza dal centroide della corsia corrente alla destinazione.
3. **Struttura** `Node`:
   - Rappresenta ogni nodo dell'algoritmo A* e include:
     - `laneId`: l'ID della corsia.
     - `cost`: il costo cumulativo del percorso.
     - `totalCost`: la somma del costo cumulativo e dell'euristica.
     - Operatore `>` per ordinare i nodi in base a `totalCost`.
4. **Funzione ausiliaria** `getLaneCentroid`:
   - Restituisce il centroide (punto centrale) di una corsia dato il suo `laneId`.
   - Essenziale per calcolare la distanza euristica.

Il **centroide** di una curva è il punto che rappresenta il centro geometrico della curva stessa. Si calcola come la media dei punti della curva campionati uniformemente. La funzione implementata come membro della classe `TrafficLane` calcola il centroide di una curva secondo i seguenti passaggi:

(a) La curva viene suddivisa in un numero prefissato di punti campionati (`samples`).

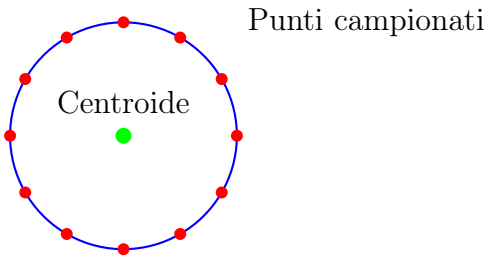
(b) Per ciascun punto campionato:
   - Si calcola il parametro t corrispondente, proporzionale alla posizione sulla curva.
   - Si ottiene la posizione del punto sulla curva tramite `pointOnCurve`.

(c) Il `centroid` è calcolato come la media delle coordinate di tutti i punti campionati.

Matematicamente, il centroide C è definito come:

$$C = \frac{1}{N} \sum_{i=1}^{N} P_i$$

dove N è il numero di punti campionati e P_i sono i punti sulla curva. Di seguito un semplice esempio grafico esplicativo:



Nell'esempio, il centroide è il punto verde al centro, calcolato come media dei punti rossi campionati lungo la curva blu.

5. **Funzione** `reconstructPath`:
   Quando viene trovato il nodo di destinazione, questa funzione:

   - Ricostruisce il percorso ottimale a partire dal nodo di destinazione.
   - Usa la mappa `cameFrom` per tracciare i predecessori.
   - Il percorso viene costruito a ritroso, partendo dalla destinazione e seguendo i predecessori, e poi invertito per ottenere l'ordine corretto.

7.2 Rispettare i limiti di velocità ed evitare le collisioni

All'interno della funzione `initFrame`, che viene chiamata ad ogni frame durante la simulazione, viene eseguito un controllo sulla posizione di ogni agente, ed in caso venga individuato un altro veicolo avanti lungo la stessa corsia viene controllata la distanza fra loro. Se la distanza è superiore al valore del parametro `distance`, inserito prima in fase preliminare della simulazione dall'utente, allora vengono considerati i limiti di velocià che vigono in quella determinata strada, accelerando in caso sia maggiore e rallentando in caso sia minore, viceversa se i due veicoli sono troppo vicini, l'ultimo rallenta frame dopo frame finchè non raggiunge una velocità sostenibile per evitare le collisioni.

7.2.1 Pseudocodice della funzione `initFrame`

La funzione aggiorna lo stato degli agenti a ogni fotogramma della simulazione. Controlla velocità, direzione, e percorso degli agenti, gestendo il raggiungimento delle destinazioni e il calcolo di nuove destinazioni.

1. **Elaborazione parallela sugli agenti**
   - Per ogni agente `agent`:
     (a) Verifica la validità dell'agente.
     (b) Recupera le seguenti informazioni dai metadati:
       – ID corsia corrente (`currentLaneId`).
       – Velocità (`frameSpeed`).
       – Posizione (`position`).
       – Destinazione (`targetPosition`).
       – Accelerazione (`frameAcceleration`).
       – Distanza di sicurezza (`safetyDistance`).
       – Percorso verso la destinazione (`destinationPath`).
     (c) Controlla se la destinazione è stata raggiunta:
       – Se la distanza tra la posizione attuale e la destinazione è inferiore a una soglia, imposta `destinationReached` a `true`.
2. **Aggiornamento della velocità e della direzione**
   - Per ogni corsia nella lista globale `m_allLanes`:
     (a) Trova la corsia corrente dell'agente.
     (b) Calcola:
       – Direzione basata sulla tangente della curva (`newDirection`).
       – Nuova velocità (`newSpeed`).
     (c) Se la destinazione non è stata raggiunta:
       – Verifica il limite di velocità e la distanza di sicurezza tramite `shouldSlowDown`.
       – Aggiorna la velocità in base alle condizioni.
     (d) Se la destinazione è stata raggiunta:
       – Riduce la velocità a zero.
       – Calcola una nuova destinazione e aggiorna il percorso.
     (e) Aggiorna i metadati dell'agente:

- Velocità.
- Direzione.

3. **Aggiornamento dell'ID della corsia**
   - Per ogni ID nel percorso (`destinationPath`):
     - Se l'ID corrente coincide con la corsia attuale e ci sono corsie successive, aggiorna l'ID della corsia.

7.2.2 Descrizione della funzione `shouldSlowDown`

La funzione `shouldSlowDown` verifica se l'agente corrente deve ridurre la velocità per evitare una collisione con altri agenti nella stessa corsia.

Funzionamento

1. Recupera la posizione dell'agente corrente.
2. Per ogni altro agente nella simulazione:
   - Se l'agente si trova nella stessa corsia:
     (a) Calcola la distanza tra l'agente corrente e l'altro agente.
     (b) Se la distanza è inferiore alla distanza di sicurezza, restituisce `true`.
3. Se nessun agente viola la distanza di sicurezza, restituisce `false`.

Capitolo 8

Guida alla simulazione e risultati

In questo capitolo verranno fornite inizialmente delle descrizioni sommarie e una guida non ufficiale riguardante l'interfaccia di Atoms ed il suo utilizzo per creare scenari urbani popolati da folle di veicoli rappresentati da folle di agenti. In secondo luogo saranno allegate delle schermate che dimostreranno i risultati ottenuti durante il lavoro di tesi.

8.1 Interfaccia di Atoms

In questa sezione verranno spiegate le funzionalità base di Atoms che sono state sfruttate durante lo sviluppo del progetto. Da notare che la seguente non è una guida in grado si sostituire la documentazione fornita da Tool Chefs, ma solo un report delle sezioni del sistema che sono state approfondite durante l'attività di tirocinio, ricordando che l'applicazione non è ancora utilizzabile e praticabile ed alcuni passaggi sono stai omessi in quanto non saranno presenti o necessari utilizzando il prodotto finale.

8.1.1 Creazione strade ed incroci

Prima di cominciare è bene avere in mente i passaggi principali per creare ed utilizzare uno scenario urbano con Atoms, ovvero per prima cosa la progettazione delle curve, poi la creazione ed il collegamento degli oggetti di Atoms ed infine la personalizzazione dei parametri per strade, incroci e agenti.
Partendo dal primo punto, non appena viene creato un nuovo progetto in Autodesk Maya sarà possibile accedere agli stumenti di creazione e manipolazione di curve, forniti dall'applicazione stessa, riportati in figura 8.1:

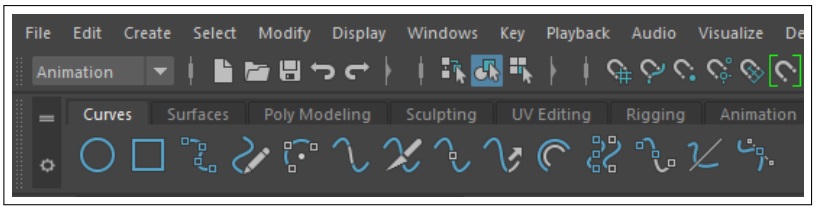


Figura 8.1: Strumenti per creare e modellare curve con Maya

Con questi strumenti è possibile creare delle nuove curve disegnandole manualmente oppure inserendo dei punti che il software collegherà a piacimento, inoltre è possibile modificare le curve già esistenti, per esempio tagliandole, spostandone i punti, aggiungendone altri ecc. Per accertarsi del corretto funzionamento della successiva generazione degli incroci, è bene sistemare le curve in modo tale che le loro estremità siano abbastanza vicine, come nel caso descritto nel caso di esempio che verrà considerato in questa sezione, in figura 8.2:

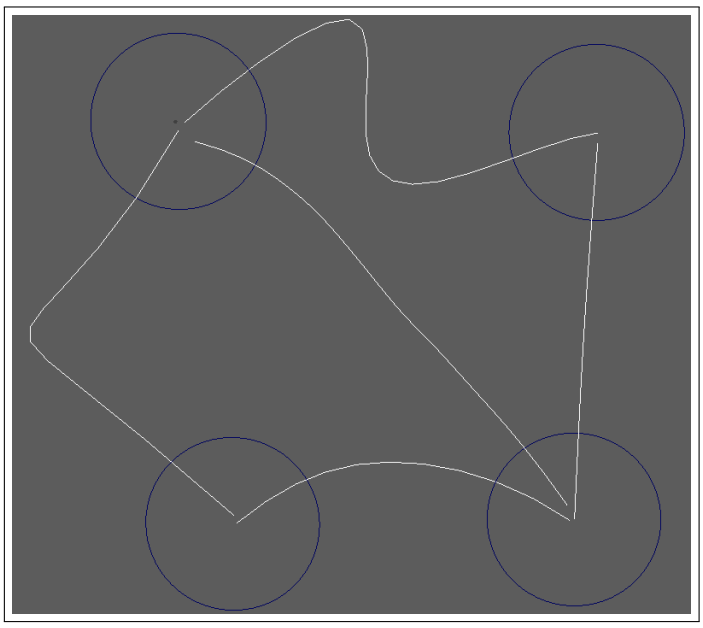


Figura 8.2: Curve create e modellate manualmente dall'utente (i cerchi rappresentano gli spazi in cui verranno creati gli incroci)

Ora che le curve sono state create, è possibile creare e collegare gli oggetti in Atoms che formeranno lo scenario urbano e che permetteranno alla logica sottostante di sfruttare i parametri che verranno in seguito inseriti per personalizzare la simulazione. Per fare ciò, è necessario accedere quello che si chiama "Node Editor", dal menu di Maya 'Windows > Node Editor', come in figura 8.3:

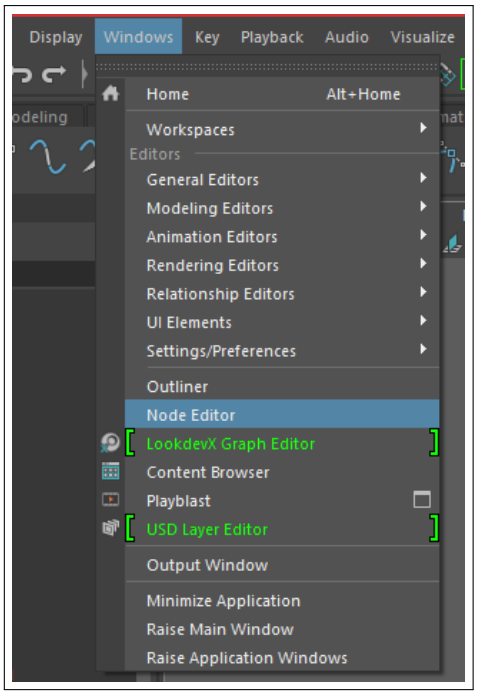


Figura 8.3: Voce del menu dalla quale è possibile accedere all'editor dei nodi

Verrà aperta una griglia vuota, nella quale è possibile trascinare gli oggetti appena creati, che saranno visibili nella voce 'Outliner' a sinistra del menu di Maya, riportato in figura 8.4:

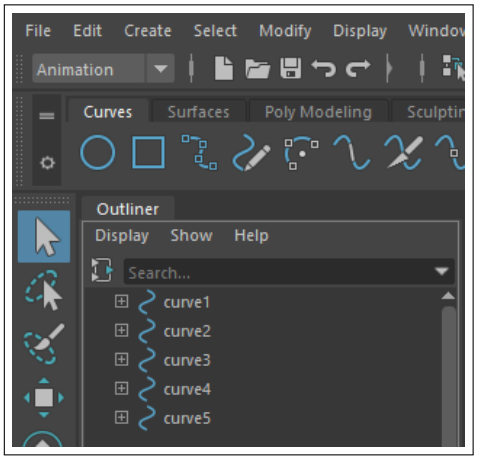


Figura 8.4: Outliner contenente le curve appena create

Ora verranno aggiunti degli oggetti di nome *curveShape* nella griglia, ognuno rappresentante una curva, questi dovranno poi essere trasformati in oggetti di Atoms, chiamati *tcTrafficStreetNode*, anch'essi uno per ogni curva, questo permette di generare nel codice l'oggetto di curva del traffico con cui lavorare; questi nuovi oggetti sono inseribile semplicemente premendo il tasto Tab e poi inserendo il nome di questi. Dopo aver creato e collegato i *tcTrafficStreetNode* sarà possibile generare gli

incroci nella medesima maniera, rappresentati da oggetti chiamati *tcTrafficCrossingNode*. Il numero di incroci dovrà riflettere le idee dello scenario che si vuole creare, nello scenario d'esempio saranno quattro, e sarà di fondamentale importanza assegnare e marcare come input le strade ai giusti incroci. Il penultimo passaggio riguarda la creazione di un nodo del traffico, chiamato *tcTrafficNode*, che farà come da wrapper per tutti gli oggetti creati precedentemente, questo perchè sarà possibile creare più nodi di traffico in un progetto solo, i quali potranno avere impostazioni ed agenti differenti. Infine l'oggetto del traffico viene collegato ad il nodo principale di Atoms, chiamato *tcAtomsNode*, che sarà l'oggetto effettivo che verrà passato al codice e che conterrà tutti i dati necessari per generare curve e agenti. Il risultato di questo lavoro all'interno del Node Editor nell'attuale caso di studio è rappresentato in figura 8.5:

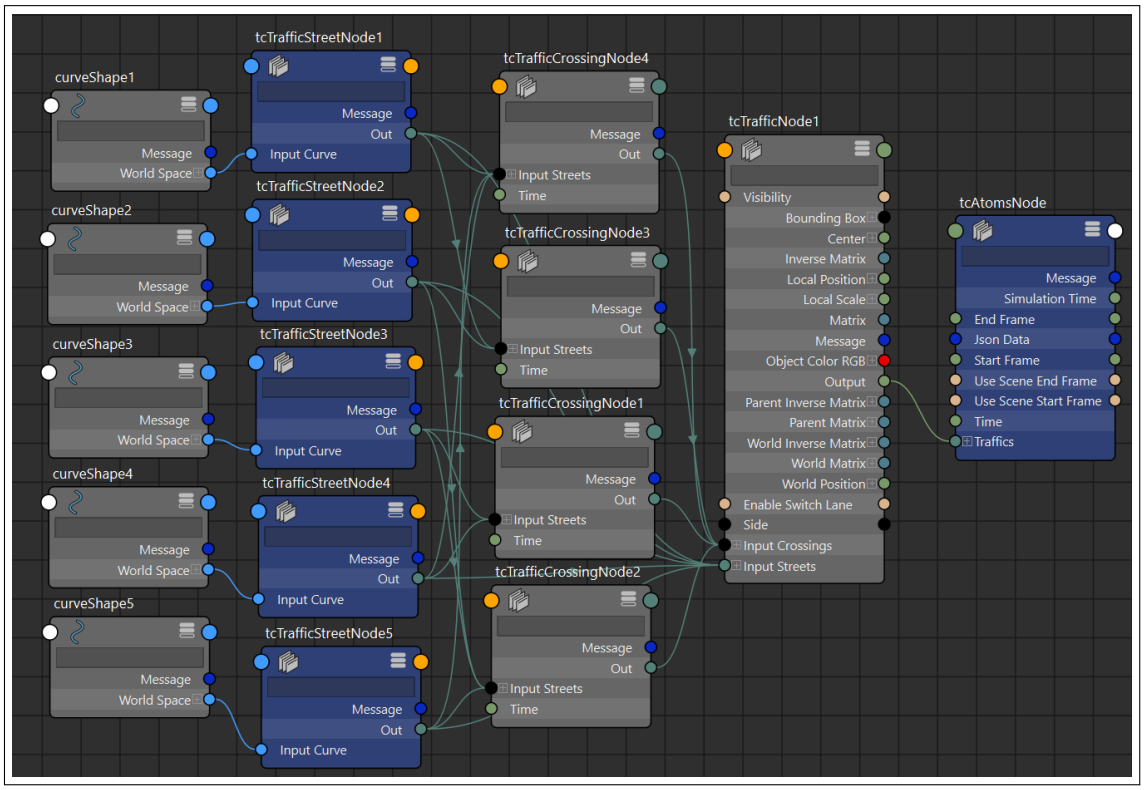


Figura 8.5: Interfaccia del Node Editor

Chiudendo l'editor dei nodi verranno già costruite le strade dipendentemente dal numero di corsie per senso di marcia inserite dall'utente, in questo caso una per senso di marcia, come in figura 8.6:

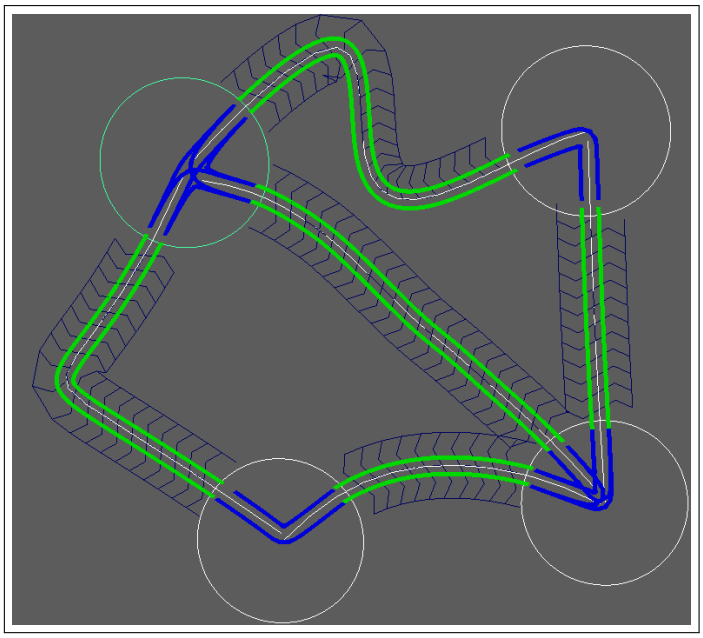


Figura 8.6: Risultato della creazione delle strade, risultato della corretta configurazione dal Node Editor

Il collegamento multiplo fra corsie in un incrocio funziona per senso di marcia, permettendo di cambiare corsia, un ingrandimento di un incrocio multiplo in figura 8.7:

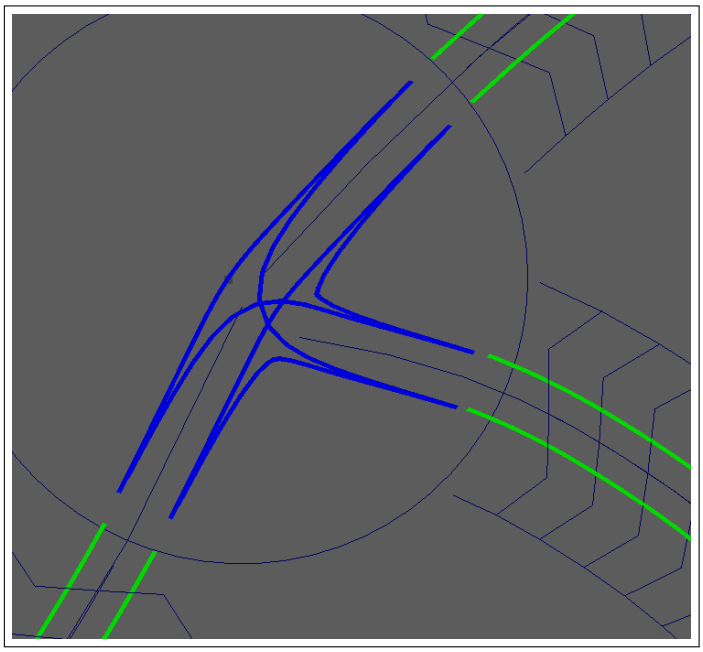


Figura 8.7: Zoom su un incrocio con tre strade entranti

Selezionando uno degli oggetti di tipo curva dall'Outliner, è possibile visualizzare e modificare dal menu di destra l'elenco dei parametri riguardanti le strade, come

in figura 8.8:

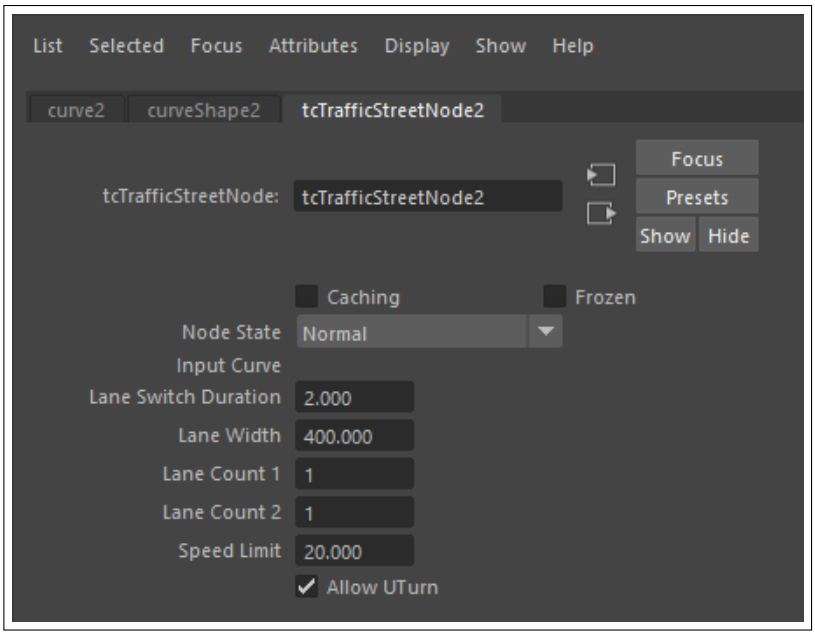


Figura 8.8: Interfaccia dalla quale è possibile modificare i parametri delle strade

8.1.2 Inserimento degli agenti

Dalla voce 'Atoms' nel menu in alto di Maya, è possibile selezionare la voce 'Create Agent Group' come in figura 8.9, il quale genererà un oggetto di tipo *tcAgentGroupNode* nell'Outliner, il quale permetterà in seguito di configurare i parametri degli agenti che popoleranno il traffico.

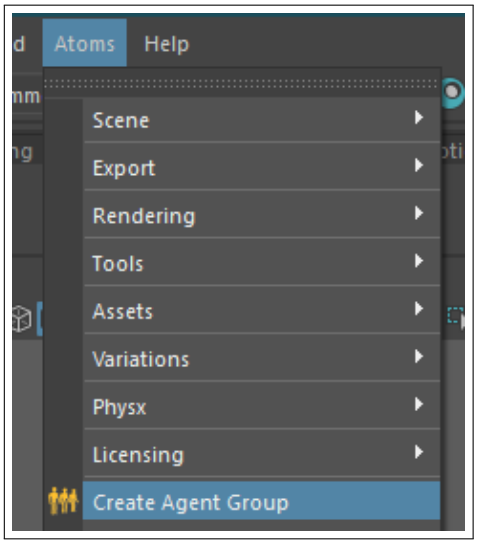


Figura 8.9: Creazione dell'Agent Group di Atoms

Ancora dalla voce ‘Atoms’ nel menu di Maya, è possibile caricare i modelli 3D (e relative animazioni) che costituiranno gli agenti di un determinato gruppo di agenti. In particolare dalla ’Atom UI’ è possibile modificare i modelli e le animazioni degli oggetti da inserire nella scena partendo dalla figura 8.10:

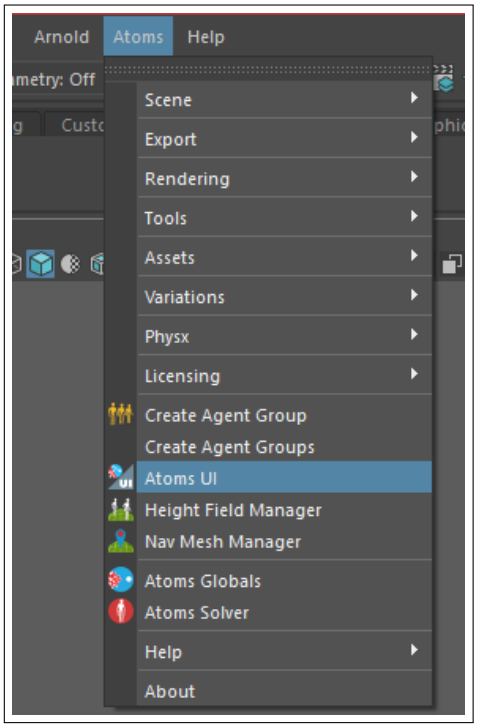


Figura 8.10: Dove si può accedere all’Atoms UI

In questo caso sono stati caricati i seguenti 4 modelli di veicoli riportati in figura 8.11:

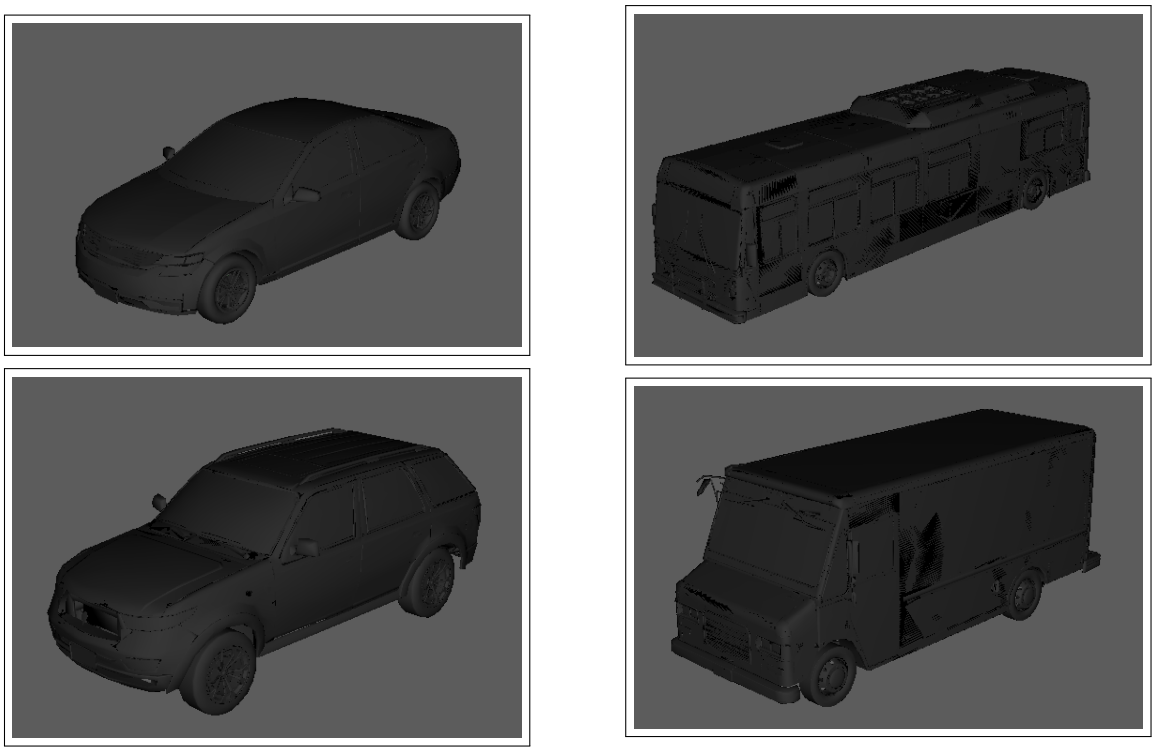


Figura 8.11: Possibili 4 modelli di macchine

Tutti i parametri che verranno presi in input dall'utente e che verranno menzionati nelle sezioni successive sono visibili e modificabili dopo aver selezionato il tcAgentGroupNode interessato nel menu di sinistra e successivamente il MyTrafficModule in quello di destra come in figura 8.12:

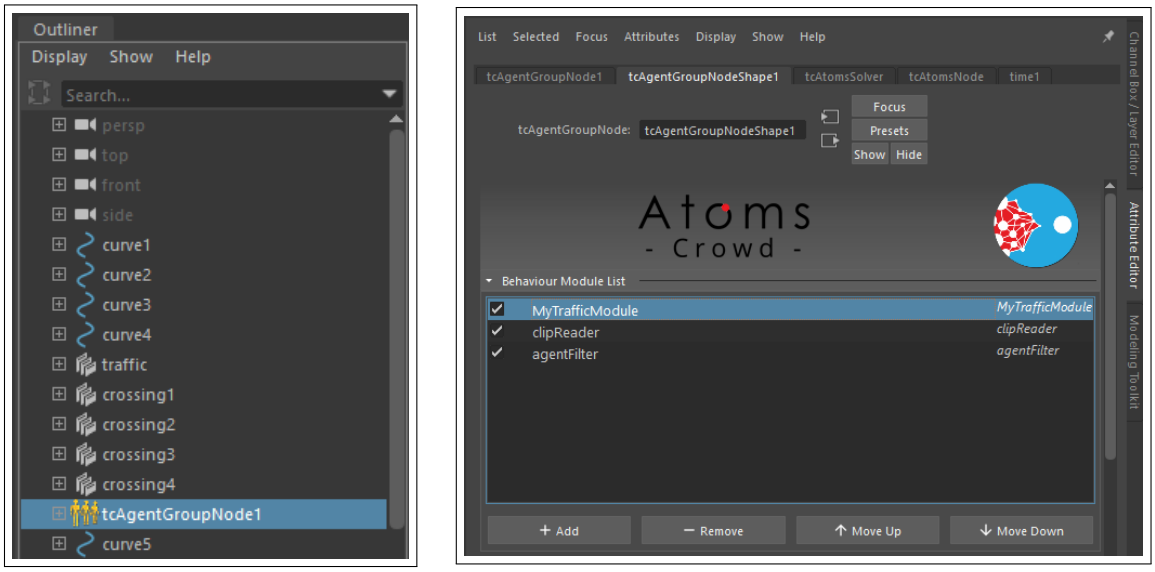


Figura 8.12: Passaggi per accedere ai parametri

Graficamente ecco in figura 8.13 come appare l'interfaccia, nella quale sono elencati alcuni dei parametri principali del traffico, come numero di agenti, tipo di agente, velocità, accelerazione, distanza di sicurezza ecc.

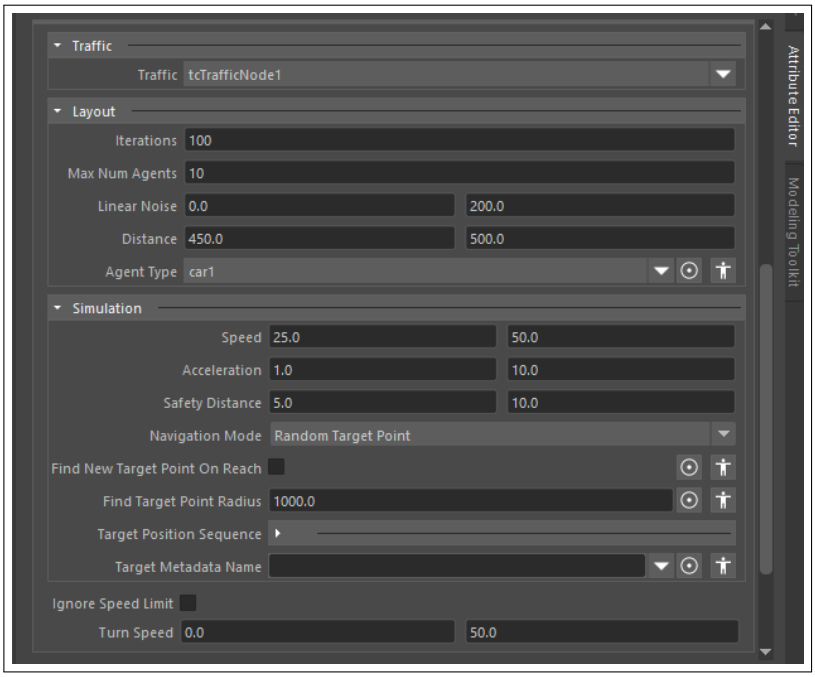


Figura 8.13: Interfaccia dalla quale è possibile modificare i parametri del traffico

Dal menu di Atoms è anche possibile, durante la simulazione, disegnare e visualizzare i metadati inseriti negli agenti, coem per esempio il loro ID, l'ID della corsia su cui su trovano ecc.. sotto la sezione 'Draw Metadata' come in figura 8.14, in cui sono stati selezionati gli ID degli agenti:

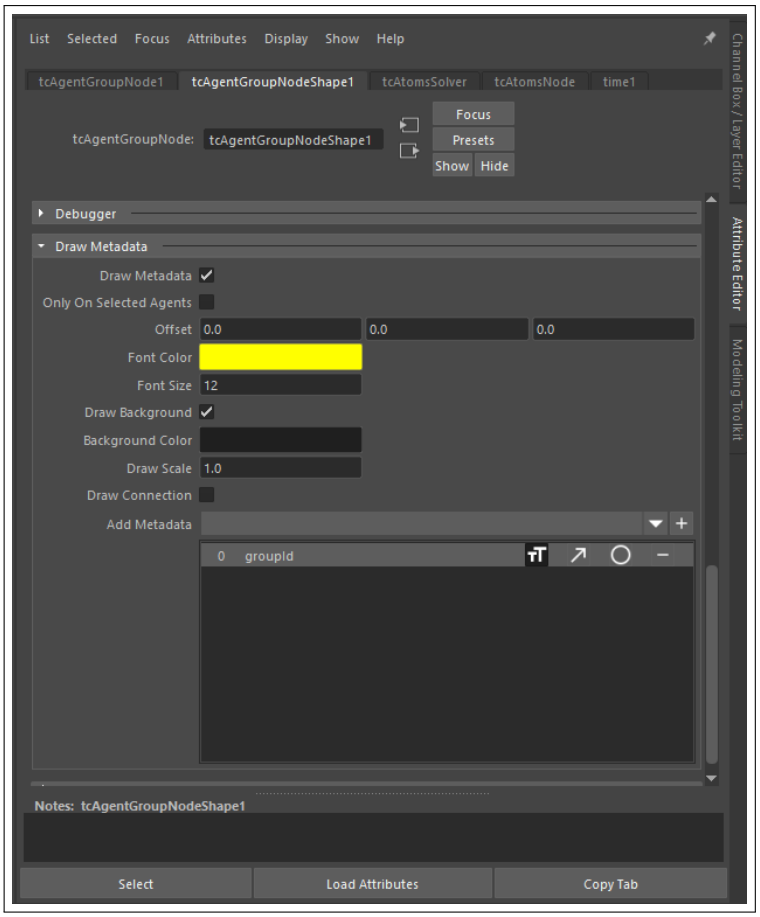


Figura 8.14: Sezione dell'interfaccia di Atoms nella quale disegnare i metadati

Nel seguente esempio in figura 8.15, a scopo di debug, sono stati inseriti in rosso tramite la funzione draw i numeri identificativi di ogni corsia e la corsia corrente di ogni agente, mentre tramite Draw Metadata sono stati inseriti i numeri identificativi degli agenti, in giallo:

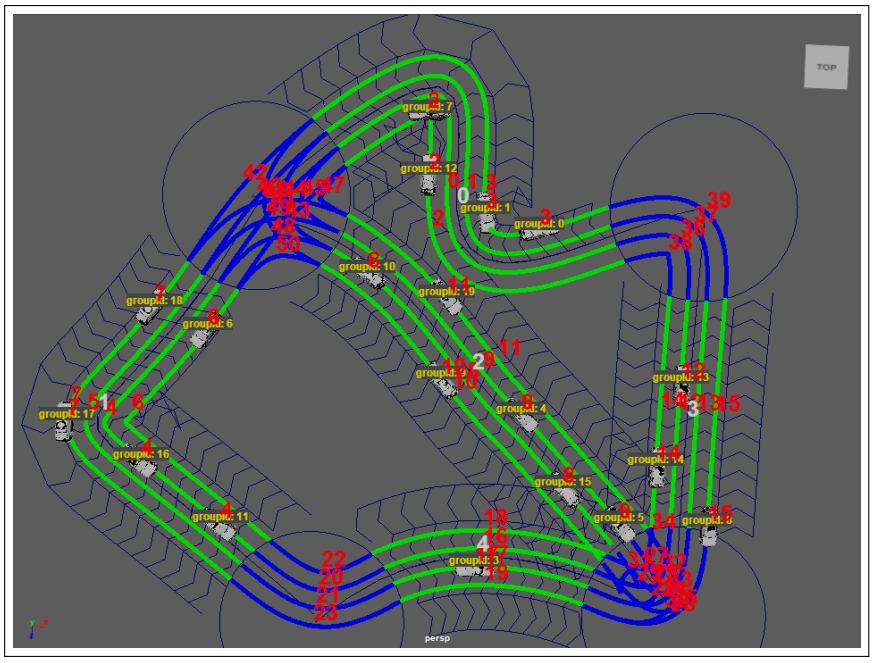


Figura 8.15: Risultato dell'aggiunta delle informazioni di debug utilizzate, come identificativi degli agenti (in giallo) e quelli delle corsie (in rosso)

8.2 Risultati

In questa sezione vengono ricapitolati brevemente i risultati principali ottenuti al raggiungimento dell'attività di tesi.

8.2.1 Corretta generazione delle corsie e posizionamento degli agenti in fase di spawn

Come si può osservare nello scenario di esempio in figura 8.16, le corsie vengono generate opportunamente in base alle indicazioni dell'utente (ovvero disegno delle strade e personalizzazione dei parametri), inoltre, viene popolato con più agenti e nessuno di essi viene generato in una posizione sovrapposta a quella di nessun altro veicolo:

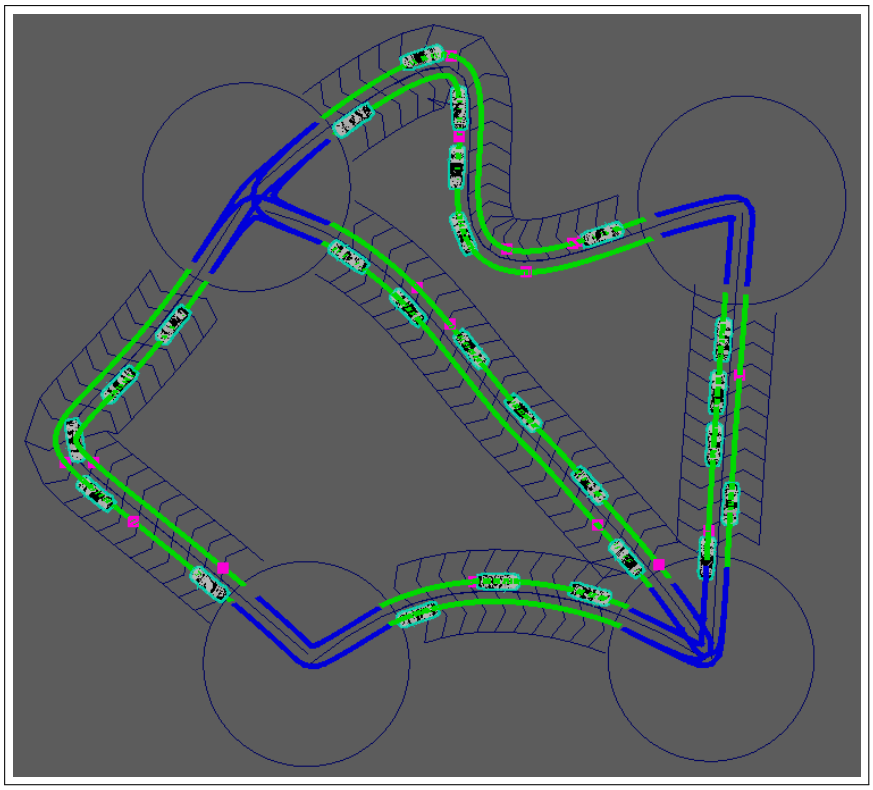


Figura 8.16: Uno scenario d'esempio popolato da più agenti

8.2.2 Mantenimento della distanza di sicurezza

Per evidenziare la distanza di sicurezza sono stati trovati, durante alcune simulazione, due agenti con dei percorsi simili, ovvero il numero 16 ed il numero 20, i quali percorrono parte della loro corsa uno dietro l'altro mantenendo la distanza di sicurezza prima di separarsi. In figura 8.17 sono stati inseriti due frame in due momenti diversi della simulazione in cui i due agenti percorrono lo scenario urbano senza sovrapporsi:

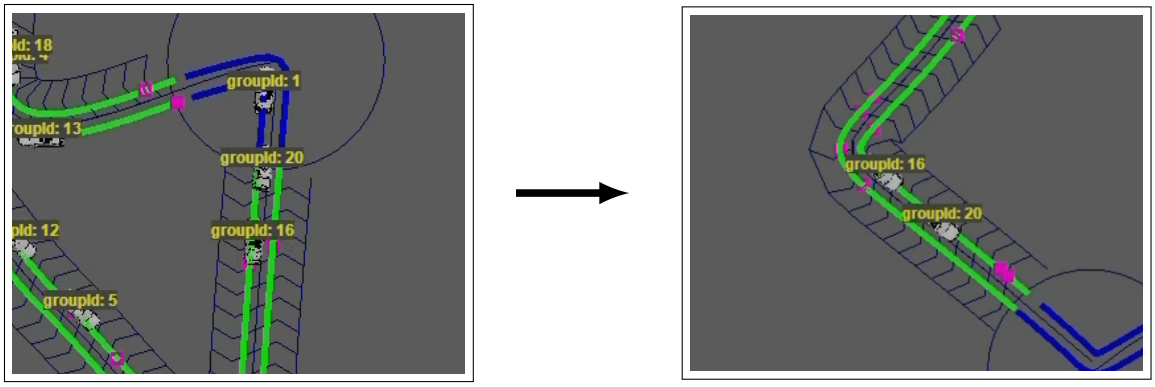


Figura 8.17: Mantenimento della distanza di sicurezza

8.2.3 Raggiungimento di una destinazione e ricalcolo della prossima

In figura 8.18 è possibile vedere nello scenario di prova con un solo agente il raggiungimento di una posizione. L'immagine in alto rappresenta la situazione di partenza ad inizio simulazione, nella quale la prima destinazione viene calcolata e, a fine della spiegazione, contrassegnata con un pallino rosa. Avviando la simulazione l'agente percorre le corsie fino al raggiungimento della posizione, momento nel quale dopo una brevissima sosta in cui viene calcolata e segnata la prossima destinazione, riparte e così via:

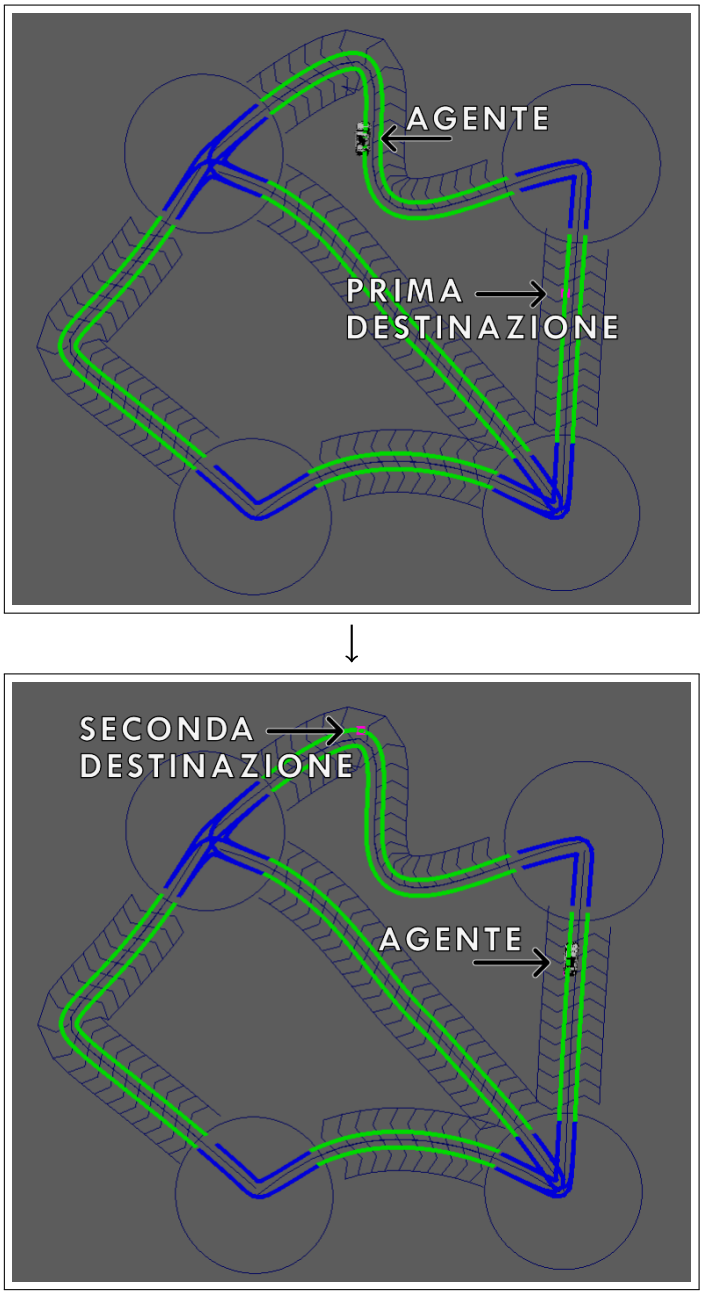


Figura 8.18: Processo di raggiungimento di una posizione con ricalcolo della prossima

Capitolo 9

Sviluppi futuri

La simulazione del traffico realizzata nel progetto di tesi, per essere realistica, deve essere migliorata sotto diversi aspetti, alcuni di questi potrebbero essere:

- innanzitutto il completamento del progetto iniziale, ovvero l'inserimento di semafori, gestione delle code, dei sorpassi, studio di migliori strategie comportamentali dei veicoli;
- strumenti per la distorsione del terreno permettendo salita e discesa dei percorsi stradali;
- strumenti per la creazione di edifici all'interno dello scenario;
- aggiunta di strisce pedonali e di pedoni, con annessi modelli grafici e logiche comportamentali;
- aggiunta di ulteriori logiche comportamentali per i veicoli per comprendere la salita e la discesa dei pedoni;
- aggiunta di parcheggi.

I precedenti sono solo alcuni dei numerosi possibili ampliamenti per Atoms, denotando una grossa espandibilità e scalabilità di questo progetto.

Referenze

[1] Documentazione di Atoms https://toolchefs.atlassian.net/wiki/spaces/ASD/pages/426056/Introduction

[2] Geometric Modeling: curves https://virtuale.unibo.it/mod/resource/view.php?id=1191820

[3] Interpolation Methods for Curve Construction https://virtuale.unibo.it/mod/resource/view.php?id=1191821

[4] "Computer Graphics: Principles and Practice" di John F. Hughes, Andries van Dam, et al.

[5] "The Illusion of Life: Disney Animation" di Frank Thomas e Ollie Johnston

[6] "Crowd Simulation" di Daniel Thalmann

[7] Three-dimensional Fuzzy Logic System for Modeling & Control – special applications of type-2 fuzzy system https://www.ieeesmc.org/newsletters/back/2009_06/SMC-HX.html

[8] Evacuation with Obstacles in Real-Time using Crowd Simulation di Tom Axblad, Álvaro González

[9] Social Force Model for Pedestrian Dynamics (1995) di Helbing, D., & Molnár, P.

[10] A Distributed Behavioral Model, SIGGRAPH 1987 di Craig W. Reynolds, Flocks, Herds, and Schools

[11] Interactive Crowd Content Generation and Analysis using Trajectory-level Behavior Learning di Aniket Bera, Sujeong Kim, Dinesh Manocha

[12] Reinforcement Learning for Multi-Agent Environments, Torrey et al.