

Intruduzione

Il CoronaVirus-19, abbreviato CoViD-19, è un virus trovato per la prima volta a Wuhan, in Cina. Esso è conosciuto anche come *Malattia respiratoria acuta*, abbreviato in **SARS-CoV-2**, dato che è un virus che come principale metodo di trasmissione utilizza le vie aeree ha avuto vita facile nel diffondersi velocemente, in effetti tutti respiriamo. In breve tempo si è passati dall'isolamento della Cina al doverlo definire **pandemia** e quindi passare ad un isolamento globale. Per la sua facilità di diffusione e per la sua pericolosità per alcuni soggetti più deboli è stato necessario modificare lo stile di vita a cui si era abituati, iniziando con i lockdown, dovendo passare alla didattica a distanza perdendo la socialità dei luoghi di studio, abbiamo visto l'economia rallentare, nei casi peggiori crollare e la socialità in presenza scomparire per via della chiusura temporanea di aziende e luoghi definiti "non di prima necessità". Questa situazione ha dato una motivazione per effettuare studi e ricerche in questo ambito, sia per ridurre i sintomi del virus, in buona parte già sta avvenendo con i vari vaccini, sia per permettere alle persone di incontrarsi e lavorare come prima della pandemia trovando metodi per ridurre i contagi come le mascherine. In Italia stiamo allentando la presa sulle restrizioni grazie anche all'avvento del *GreenPass* il quale permette di certificare che si è vaccinati o che si hanno gli anticorpi dovuti ad un'infezione da CoViD-19. Questo però non ferma il contagio dato che il vaccino permette agli individui di avere sintomi più lievi ma è comunque possibile essere contagiati e diffondere il virus. Uno degli aspetti che coinvolge un po' tutte le persone, sia per divertimento che per lavoro è il trasporto pubblico. Non tutti hanno un mezzo di trasporto proprio o per molti addirittura non conviene, basti pensare alle grandi città con lunghe code per il traffico a tal punto da fare prima a far il tragitto a piedi. Quindi molti optano per i trasporti pubblici ma qui è dove avvengono i principali contagi dovuti ai luoghi chiusi, stretti e poco arieggiati che impediscono di seguire perfettamente le

normative dettate dal governo e dall'Organizzazione Mondiale della Sanità (OMS); questo è uno dei principali motivi del lavoro di tesi.

L'obiettivo del progetto di tesi è quello di creare una simulazione ad agenti estendibile a diversi tipi di percorsi e abitudini cittadine diverse, da cui recuperare dati sulle persone e sui contagi che avvengono su un mezzo di trasporto pubblico (nel nostro caso si tratta di un autobus). La simulazione è stata realizzata interamente in Unity3D, un motore grafico che consente lo sviluppo di contenuti interattivi digitali, sia 2D che 3D, come ad esempio videogiochi e simulazioni. Sono stati anche utilizzati tool ed algoritmi per risolvere vari problemi, come ad esempio abbiamo utilizzato *A* Pathfinding Project* a cura di Aron Granberg per il calcolo del percorso degli agenti.

Gli agenti vengono distinti in:

- Sano, indicato in verde
- Contagioso, indicato in rosso
- Infetto, indicato in giallo

Come è visibile nella Figura 1

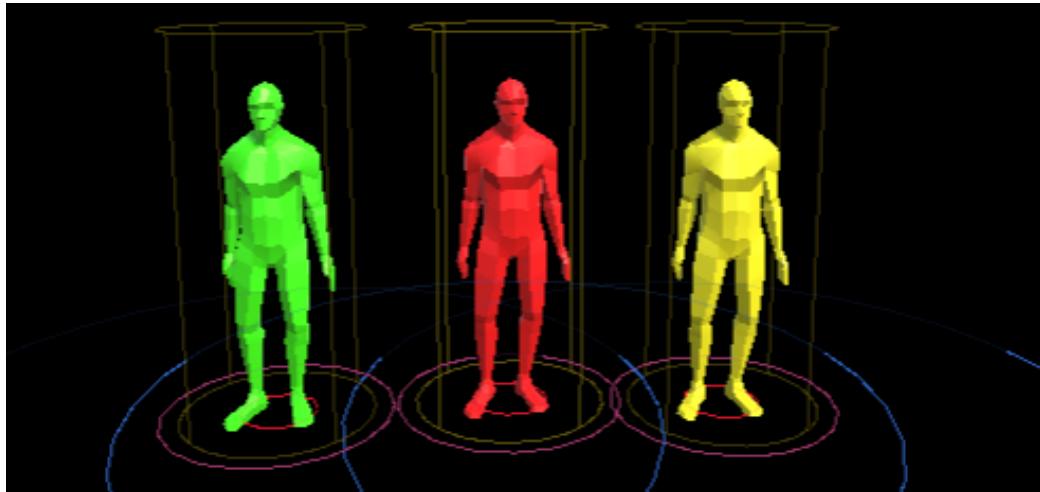


Figura 1: Stati degli agenti

I dati sul numero di pendolari della tratta sono stati presi da un articolo riguardante uno studio pre-pandemia sui viaggiatori della cittadina giapponese di Obuse[4], della quale è stato preso il percorso della tratta del bus.

La simulazione permette di selezionare la percentuale di contagiati iniziale e la percentuale di infezione del virus in modo tale da effettuare facilmente diversi test. Il numero di persone alle fermate è selezionabile per ogni singola fermata in modo tale da poter essere utilizzate in diversi percorsi o più semplicemente applicarlo alle stime dei pendolari attuali.

All'avvio della simulazione viene generato un file .csv che verrà riempito per ogni tratta del bus con le informazioni generate sugli agenti, come il numero i agenti totali, quelli contagiosi, quelli infetti e quelli sani.

La tesi sarà suddivisa nelle seguenti sezioni:

- **Stato dell'arte:** verrà mostrato lo stato delle ricerche sulla diffusione del CoViD-19 in ambito di trasporti pubblici, citando elaborati e mostrandone i punti di forza e le limitatezze.
- **Presentazione delle tecnologie:** in questa sezione verrà mostrato in dettaglio le tecnologie utilizzate per la creazione della simulazione.
- **Simulazione nel dettaglio:** verrà mostrato il lavoro svolto per la realizzazione della simulazione mostrando in dettaglio i vari oggetti, come sono stati utilizzati i tool e il lavoro svolto senza tool.
- **Risultati ottenuti:** verrà mostrato i risultati ottenuti da vari test della simulazione.
- **Conclusione e sviluppi futuri:** si analizzeranno le limitazioni del lavoro allo stato attuale e si metteranno in risalto idee per sviluppi futuri

Capitolo 1

Stato dell'arte

Dalla scoperta del salto di specie effettuato dal CoViD-19 sull'essere umano ad oggi ci sono stati innumerevoli studi, ricerche su vari aspetti della malattia, sul suo impatto sulla società in ambito politico, sociale ed economico. Ci sono anche varie simulazioni sulla sua diffusione in varie situazioni come trasporti pubblici, uffici, ospedali, locali pubblici.

Come primo articolo si parlerà di "Simulaion-based Estimation of the Spread of COVID-19 in Iran" [2] pubblicato il 27 Marzo 2020 su medRxiv. Come primo luogo l'articolo mette in luce i problemi dovuti ai dati provenienti dai vari paesi colpiti dal virus, definendoli altamente inaffidabili perché non tengono conto che i sintomi lievi del CoViD-19 sono assimilabili al raffreddore stagionale o alla comune influenza, in più molti testi vengono effettuati tramite screening limitati, un metodo per effettuare esami a tappeto allo scopo di individuare una malattia. Tutti questi dati non vengono combinati con le statistiche ufficiali sulla diffusione del virus e si va quindi a sottovalutare la quantità di infetti. Lo studio è stato effettuato sulla diffusione del virus in Iran nei primi mesi del 2020.

È stato sviluppato un modello dinamico per fornire un quadro affidabile dello stato della malattia basandosi sui dati esistenti. Il modello si basa sul framework **SEIR**(Suscettibile, Esposto, Infetto, Ricoverato);

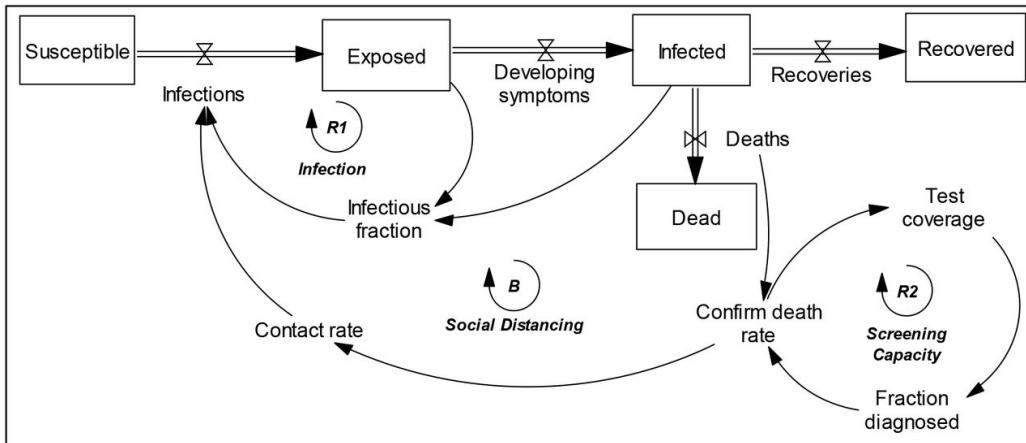


Figura 1.1: Questo è la rappresentazione di un modello SEIR

I modelli matematici, in epidemiologia, sono modelli simbolici costituiti da una o più equazioni che considerano diversi parametri per prevedere l'andamento di una malattia in diverse condizioni ambientali o per calcolare il rischio di morte o l'aspettativa di vita nel corso di una pandemia di specifiche entità. SEIR è uno di questi modelli matematici, esso considera il periodo di incubazione durante il quale un individuo è infetto ma non contagioso, in questo caso si trova nello stato di *"Esposto"*, questo è il principale motivo per cui è stato scelto questo modello, tenendo conto il periodo di incubazione del CoVid-19 che ha un massimo di 14 giorni.

Come primo parametro è stato registrata la frequenza di contatti medi tra persone man mano che venivano segnalati i decessi, poi sono stati differenziati i casi segnalati dai casi effettivi e di come questa differenza cambia con l'andare dell'epidemia(all'epoca ancora definibile epidemia). Nel modello sono stati utilizzati sia dati ufficiali che dati stimati dalla comunità medica in modo tale da ricostruire un quadro più completo della situazione epidemica. Questi dati vengono raccolti per poi passare ad un simulazione con metodo *Monte Carlo*; è una classe di metodi computazionali basati sul campionamento casuale per ottenere risultati numerici, questo metodo è utilizzato per trarre stime attraverso simulazioni. Da quest'analisi sono state trovate sei possibili situazioni della diffusione della malattia in relazione ad effetti stagionali e misure di distanziamento sociale.

I risultati dello studio mostra che il vero numero di casi probabilmente è molto più grande di quello effettivamente registrato, quasi il doppio degli

infetti. Lo studio avverte sul fatto che i dati potrebbero dichiarare molti meno casi di quelli che sono nella realtà portando la popolazione e i governi a sottovalutare la situazione.

Il prossimo documento di cui si parlerà è *"How simulation modelling can help reduce the impact of COVID-19"*[1] a cura della professoressa Christine Currie pubblicato dal CORMSIS, centro di ricerca operativa e statistica dell’Università del Southampton, il 15 Aprile 2020 su Taylor&Francis Online il quale tratta dei vari modelli utilizzati dai governi e dall’Organizzazione Mondiale della Sanità per decidere le migliori strategie per debellare la minaccia del CoViD-19. I modelli utilizzati sono modelli epidemiologici volti a comprendere la diffusione della malattia. In quest’articolo verranno discussi di come i vari modelli di simulazione potrebbe aiutare a prendere decisioni. Il presente articolo presenta due obbiettivi, il primo è dare una guida ai diversi modelli di simulazioni e quali tipi possono essere utili a prendere decisioni durante un’emergenza sanitaria; il secondo è una *"Chiamata alle armi"* dei modellatori di simulazioni per migliorare le ricerche. I modelli di tipo epidemiologico sono un’idea per predire il numero di nuovi casi identificando le migliori misure per ridurre i contagi, ma non aiutano ad organizzare l’organizzazione delle terapie intensive all’interno degli ospedali. Come nel paper precedente anche nel presente si indica il modello *SEIR* come il modello più popolare per descrivere l’attuale pandemia per via del tempo di incubazione dato dallo stato di *"Exposed"*.

Questo paper considera quattro metodi di modellazione:

- *Sistemi dinamici*: è un modello basato su equazioni differenziali che rappresentano azioni nel mondo.
- *Modello basato ad agenti* può essere utilizzato per modellare l’interazione tra gli individui. In questi modelli utilizzati per monitorare la diffusione di una malattia è molto importante gestire le reti sociali e i movimenti nello spazio.
- *Simulazione ad eventi discreti*, il sistema è rappresentato, nella sua evoluzione nel tempo, con variabili che cambiano istantaneamente il loro valore in ben definiti istanti di tempo appartenenti ad un insieme numerabili, questi istanti sono definiti eventi.
- *Simulazione ibrida*: sono modelli che combinano due o più tecniche di modellazione citate precedentemente.

Dato che le simulazioni sviluppate con i metodi descritti in precedenza possono diventare molto complesse e richiedere tempo per l'esecuzione sono state sviluppate le simulazioni distribuite, le quali permettono di collegare più simulazioni in una singola in scala maggiore permettendo una velocità maggiore.

Di seguito il presente paper introduce le tipologie di decisioni che sono state prese grazie all'utilizzo di simulazioni durante la pandemia da CoViD-19; esse vengono divise in tre sezioni:

- *Decisioni che hanno avuto effetto sulla diffusione del virus.*
- *Decisioni che riguardano la gestione delle risorse.*
- *Decisioni sulla salute della popolazione.*

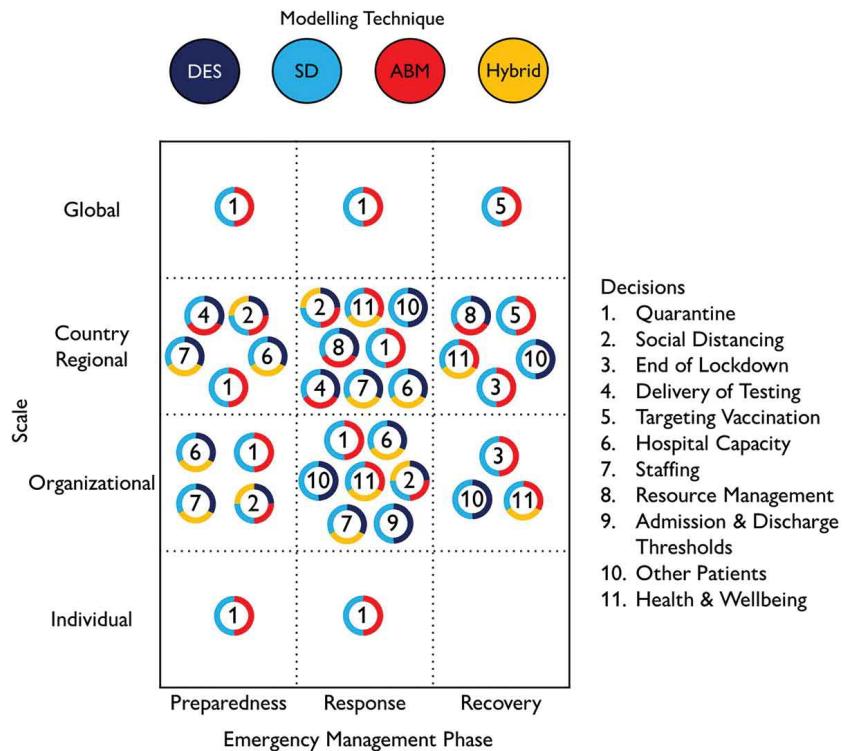


Figura 1.2: Questo è uno schema sulle decisioni consigliate dal risultato di diverse tipologie di modelli per le simulazioni.

La Figura 1.2 mette in luce le diverse decisioni suggerite dalle quattro tipologie di modelli in base a scala organizzativa e fase di emergenza. Ora andiamo più nel dettaglio nelle decisioni per evitare la diffusione del virus descritte nel presente paper.

Come prima decisione possibile abbiamo quella della *quarantena*, questo per evitare del tutto la diffusione del virus. Questa decisione è d'obbligo una volta individuato un caso di corona virus, questo fino alla sua completa guarigione in modo tale da non poter contagiare nessuno. L'articolo cita un articolo del 2016 intitolato "The Ebola crisis and the corresponding public behavior: A system dynamics approach" il quale dimostra di come la messa in quarantena dei casi di ebola ha avuto un impatto importante su combattere la malattia; questo però non è applicabile così facilmente alla pandemia da corona virus dato che i sintomi dell'ebola sono ben visibili sulla pelle dei malcapitati, a differenza di quelli da CoViD-19 che possono anche essere lievi o assenti.

Una seconda decisione che ha avuto un grande impatto sul nostro stile di vita, ma allo stesso è stato di grande aiuto per combattere la diffusione del virus sono le misure di *distanziamento sociale*. Per distanziamento sociale si intendono quei comportamenti nel quale riduciamo, in parte ,il contatto umano in modo tale da essere sicuri di essere contagiati o di contagiare e quindi puntare a ridurre la crescita della curva dei contagi. Esempi di distanziamento sociale possono essere:

- mantenere una distanza di sicurezza di circa due metri in modo tale da non far arrivare gocce di saliva alla persona con cui stiamo colloquiando
- lavorare da casa quando è possibile, infatti molte aziende sono passate allo smart-working, in modo tale da continuare a lavorare tranquillamente anche da casa.
- Cancellare eventi i quali prevedono un elevato numero di persone.
- chiudere le scuole e optare per le lezioni in remoto.
- evitare di incontrare fisicamente amici e parenti.

Un decisione molto più drastica delle due precedenti è il lock-down imposto dal governo in modo tale da costringere la popolazione al distanziamento sociale e a lasciare le abitazioni sono in caso di prima necessità, come andare a fare la spesa, andare a lavorare ed uscire per motivi di salute. Per ovviare a

problemi legati all'economia ogni governo può includere una soglia di valori sul quando rilassare le restrizioni in modo tale da permettere ai lavoratori che non eseguono un lavoro considerato di "Prima necessità" di lavorare facendo girare l'economia.

In conclusione il documento lascia un appello definendolo "The call to arms" invitando a sviluppare simulazioni in merito alla diffusione e ai comportamenti per combattere la diffusione del virus in modo tale da supportare la ricerca, per poi fare l'esempio dei "codebreakers" britannici insieme ad Alan Turing, durante la Seconda Guerra Mondiale, decriptarono Enigma mostrando che l'unione di un buon team risolve problemi difficili.

Lo sviluppo delle simulazioni si è dimostrato essere molto efficace, basti pensare al fatto che questo paper è stato redatto il 27 Marzo 2020 e seguendo le decisioni suggerite dalle simulazioni stiamo piano tornando alla normalità.

Esistono numerosi studi che utilizzano simulazioni riguardo al CoViD-19 per varie motivazioni, quali: allo scopo di fornire un modello per comprendere al meglio le dinamiche spazio-temporali della diffusione della pandemia [5]; oppure allo scopo di predire i rischi nel rimuovere le limitazioni di movimento alla popolazione illustrando anche l'efficacia delle mascherine in ambienti di simulazioni a lungo termine.[6]; o ancora studi con dati meno accurati ottenuti con sondaggi telefonici durante le prime fasi di pandemia per avere dati aggiuntivi rispetto a quelli forniti da istituzioni governative. [8].

Capitolo 2

Presentazione delle tecnologie

In questo capitolo verranno mostrate, nel dettaglio, tutte le tecnologie utilizzate per la realizzazione della simulazione.

2.1 Unity

In primo luogo abbiamo la base della simulazione, Unity[7], un motore grafico multipiattaforma sviluppato da Unity Technologies utilizzato per lo sviluppo di contenuti interattivi, quali: videogiochi; simulazioni; esperienze interattive con dispositivi di realtà aumentata.

Possiamo dividere Unity in due sottosistemi, Unity Editor e Unity Engine.

2.1.1 Unity Editor

Con Unity Editor si intende l'interfaccia grafica di Unity, la quale è costruita per rendere facili le interazioni da parte dell'utente, essa è completamente modificabile a piacere dell'utente che la possibilità di salvare il suo editor in profili per ogni situazione.

Ora parleremo delle diverse componenti della UI dell'editor di Unity.

Inspector

Ogni oggetto in unity è descritto dal concetto di GameObject, il quale identifica ogni oggetto in Unity. Ogni GameObject è decorato dai cosiddetti *Components*, i quali possono essere di diverso genere e comprendono mesh,

script(in C#), animazioni, rigidBody(il quale gestisce le componenti fisiche di un GameObject) e molti altri.

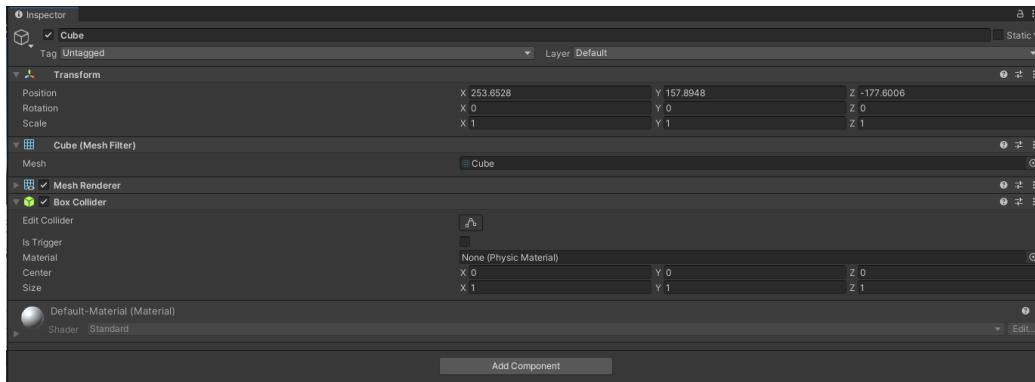


Figura 2.1: Questo è l'inspector di un semplice cubo in Unity

Come è possibile vedere nella figura 2.1 l'inspector permette di aggiungere, rimuovere e modificare i *component* dei GameObject.

Hierarchy

L'Hierarchy contiene tutti gli oggetti di scena, come si può intuire dal nome, gli oggetti vengono mostrati tramite una gerarchia ad albero dove come radice abbiamo la *scena*.

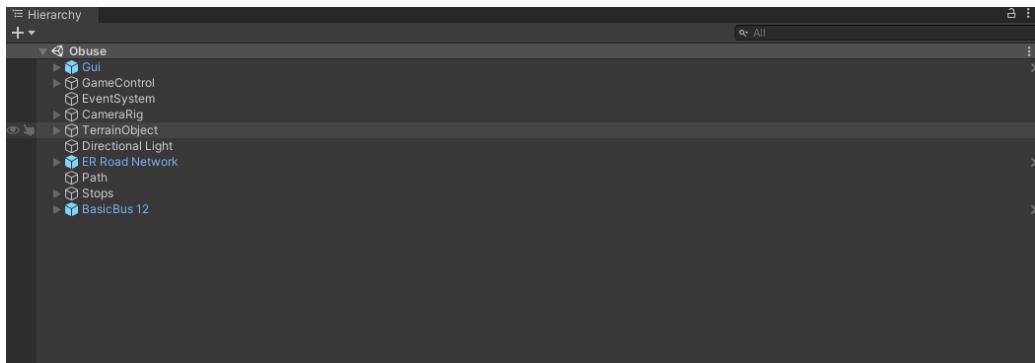


Figura 2.2: Questa è l'Hierarchy della simulazione.

Come si può vedere dalla figura 2.2 possiamo vedere come radice abbiamo l'oggetto "Obuse" che è la scena principale della simulazione e come figli ci sono tutti gli oggetti di scena.

Project

La finestra *Project* permette di gestire la directory del progetto nella quale sono presenti script, prefab, modelli, texture, file audio e scene.

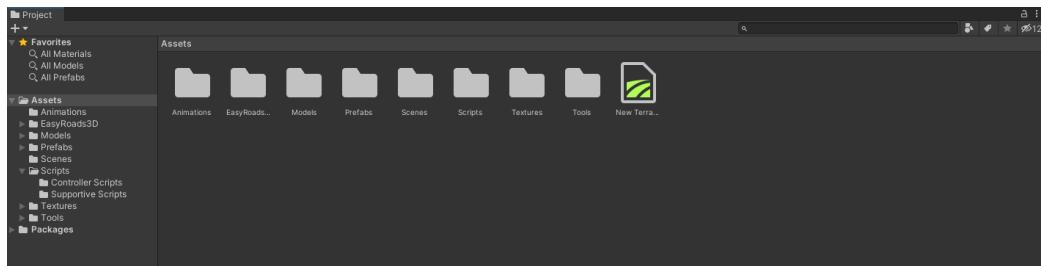


Figura 2.3: Questa è la finestra Project della simulazione

Come è possibile vedere nella figura 2.3 il progetto è diviso in diverse cartelle per avere un'organizzazione migliore delle risorse.

Scene

La finestra "*Scene*" rappresenta la scena corrente, tutti i *GameObject* che sono nell'*Hierarchy*, come detto precedentemente, sono nella scena corrente. Dalla scena è possibile manipolare i *GameObject* con strumenti di traslazione, rotazione e ridimensionamento. Tramite il mouse e la tastiera possiamo navigare con facilità nella scena oppure per essere più rapidi è possibile fare focus sull'oggetto selezionato, o dall'*Hierarchy* o dalla scena, premendo il tasto *F*.

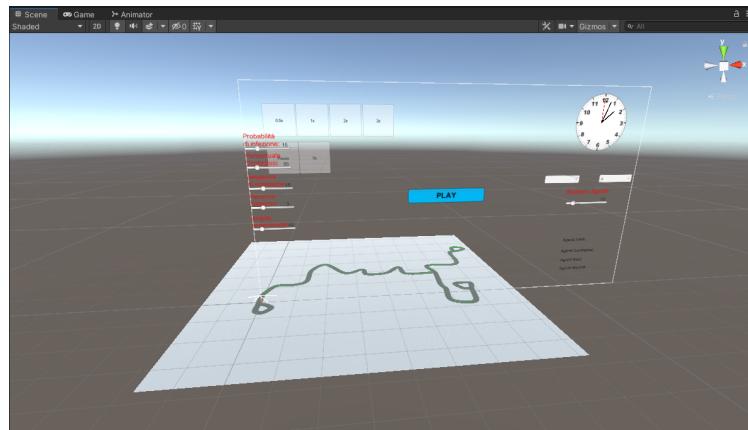


Figura 2.4: Questa è la scena della simulazione

Dalla figura 2.4 possiamo vedere tutta la scena della simulazione, compresa di gui(Delineata dal quadrato con bordi bianchi)

Game

La finestra *Game* viene utilizzata quando si esegue il progetto, si passa alla finestra di game con la quale è possibile effettuare testi direttamente da Unity. Allo stesso tempo è possibile modificare parametri dalla scena e dall'Inspector mentre il Game è in esecuzione.

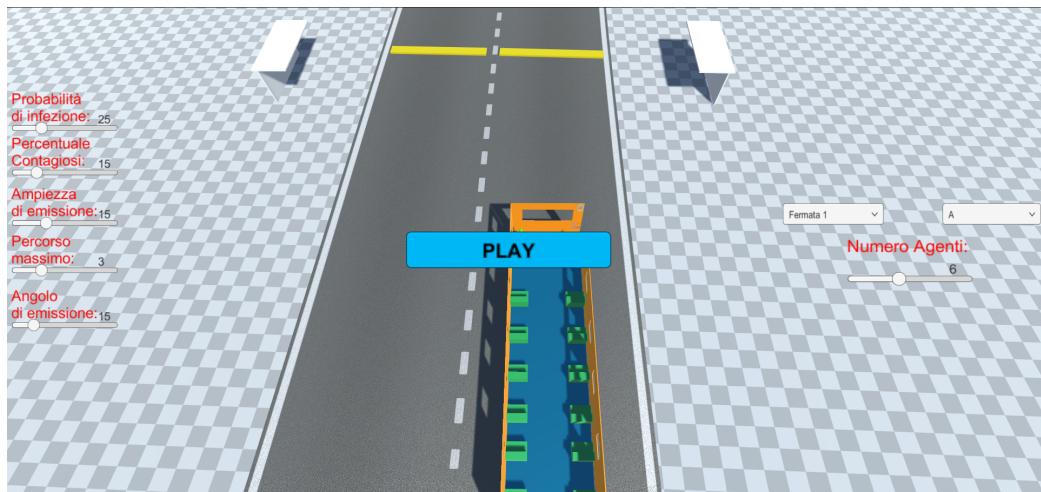


Figura 2.5: Questo è il game in pausa

Nella figura 2.5 viene mostrato il game quando è in pausa prima di iniziare, a sinistra si possono vedere vari slider per la gestione dei parametri della simulazione, mentre a destra abbiamo dei menù dropdown per la selezione degli agenti nelle varie fermate.

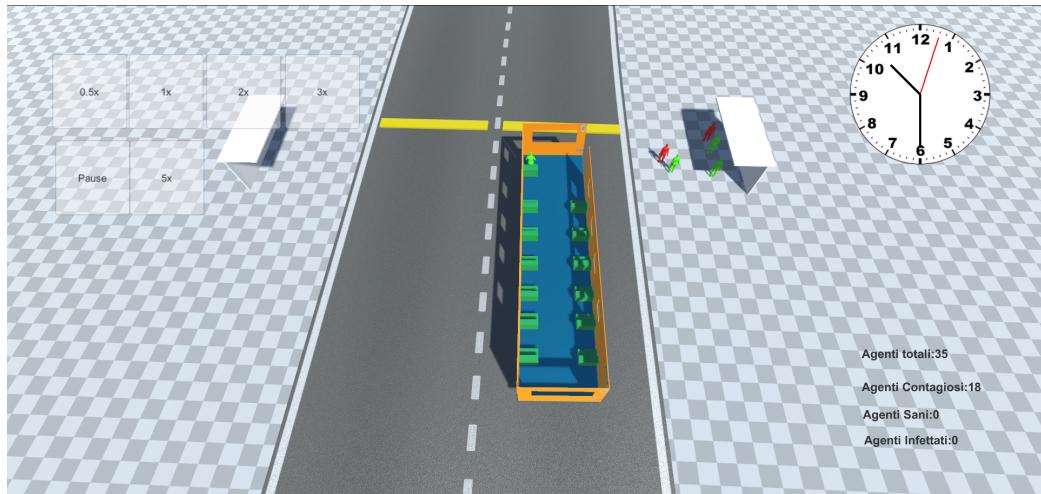


Figura 2.6: Questo è il game una volta iniziato

Nella figura 2.6 viene mostrato il game in play, quindi cambia l'interfaccia, a sinistra abbiamo i vari bottoni per la gestione della velocità della simulazione, in alto a destra abbiamo l'orologio che scandisce il tempo della simulazione; il quale rispetta il tempo reale della tratta di Obuse e infine, in basso a destra, abbiamo i dati sugli agenti.

Animation e Animator

Come è stato detto in precedenza, gli agenti della simulazione sono completamente animati, quindi ora parleremo di Animazioni e Animator. La finestra Animator permette di creare e modificare le animazioni tramite la timeline dell'oggetto da animare.

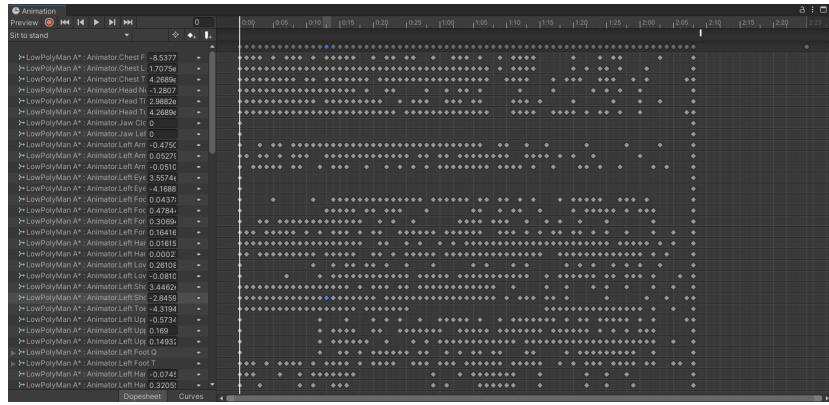


Figura 2.7: Questa è la finestra Animation dell’animazione ”*Sit to Stand*” dell’agente che gli permette di sedersi sull’autobus.

Come è possibile vedere dalla figura 2.7 sulla sinistra abbiamo le sezioni dell’agente da animare, in alto abbiamo la timeline e al centro abbiamo i frame corrispondenti alle sezioni in corrispondenza del frame nella timeline. Tutte le animazioni sono gestite dal componente *Animator*, una macchina a stati finiti in grado di gestire differenti stati e passare da uno stato all’altro attraverso transizioni.

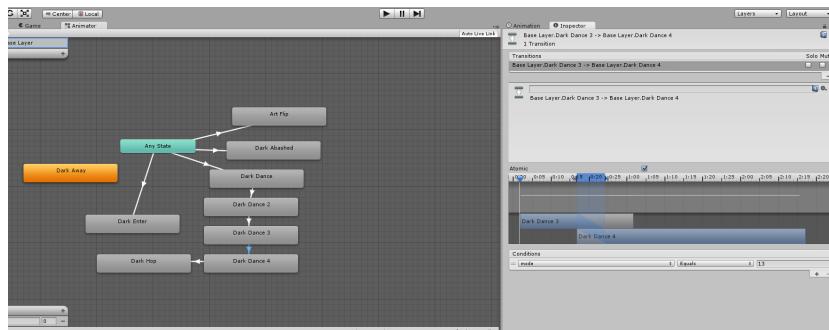


Figura 2.8: Questo è il component animator.

Dalla figura 2.8 è possibile vedere, sulla sinistra, la macchina a stati finiti che è l’*animator*; sulla destra invece abbiamo il focus di una transizione, più precisamente dallo stato *Dark Dance 3* a *Dark Dance 4* la quale avviene quando la variabile *mode* ha valore pari a 13, come è possibile vedere nella sezione *conditions*.

2.1.2 Unity Engine

Abbiamo parlato della parte di interfaccia utente di Unity, ora parleremo di quello che c'è dietro la realizzazione di un applicativo in Unity. Il motore di Unity permette di gestire la fisica del mondo virtuale in modo semplice aggiungendo vari componenti ai gameObject, come il Rigidbody o i Collider. Più nello specifico il Rigidbody è un componente che permette ai GameObject di agire in risposta alle forze del mondo di gioco; ad esempio è possibile indicare se un oggetto deve essere soggetto alla gravità oppure è possibile bloccare rotazione o traslazione su uno specifico asse, tutto questo è possibile selezionarle all'interno dell'Inspector o tramite scripting in C#.

I Collider, invece, permettono di gestire la collisione tra gli oggetti, più nello specifico permette di ascoltare le collisioni che avvengono tra Collider e Rigidbody. I Collider hanno diverse forme per addattarli ad ogni situazione; ad esempio abbiamo il BoxCollider, Sphere Collider, MeshCollider, CapsuleCollider.

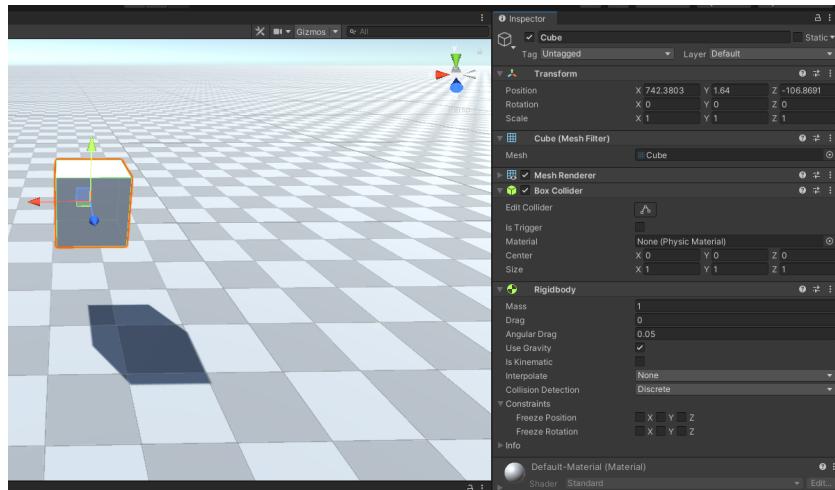


Figura 2.9: Questo è un cubo in unity, a destra possiamo vedere l'inspector con i componenti Box Collider e Rigidbody.

Nella figura 2.9 possiamo vedere un semplice cubo in unity e sulla destra il suo Inspector. Il component collider presenta parametri riguardo grandezza e posizione, e un campo *"Is Trigger"* il quale indica che il collider è un trigger, cioè è attraversabile ed è possibile intercettare entrata, presenza all'interno e

uscita di un oggetto. Mentre il Rigidbody presenta parametri per la massa, l'attrito e i vincoli sui movimenti e rotazione.

Telecamera

La camera in unity è uno dei GameObject più importanti, essa presenta due componenti principali:

- **Camera Component**, ciò che permette alla camera di riprendere il mondo di gioco
- **Audio Listener**, è il nostro "orecchio sul mondo di gioco" infatti ci permette di catturare i suoni prodotti dalle fonti sonore.

Il render della telecamera è gestito da una particolare figura geometrica chiamata *frustum*, nel caso specifico della telecamera è definito frustum di visione, il quale permette per l'appunto, di vedere quello che è presente in esso. Ogni parametro del frustum è modificabile nel *Camera component*.

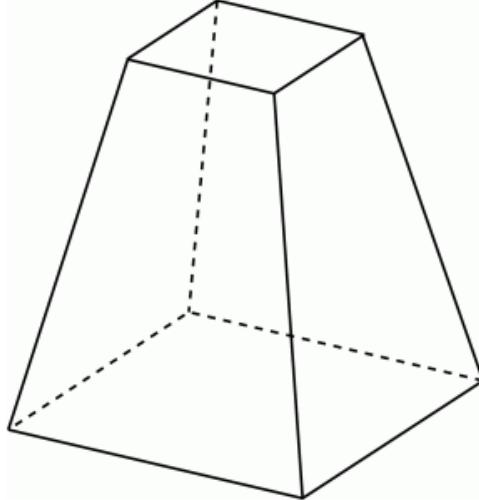


Figura 2.10: Questa è la figura geometrica frustum

Particle System Shuriken Unity presenta diversi sistemi di particelle, quello utilizzato per la simulazione è il *Particle System Shuriken*, il quale permette di gestire facilmente la collisione delle particelle. Il

sistema di particelle permette di creare molti effetti tramite l'utilizzo di moduli per la gestione dei parametri.

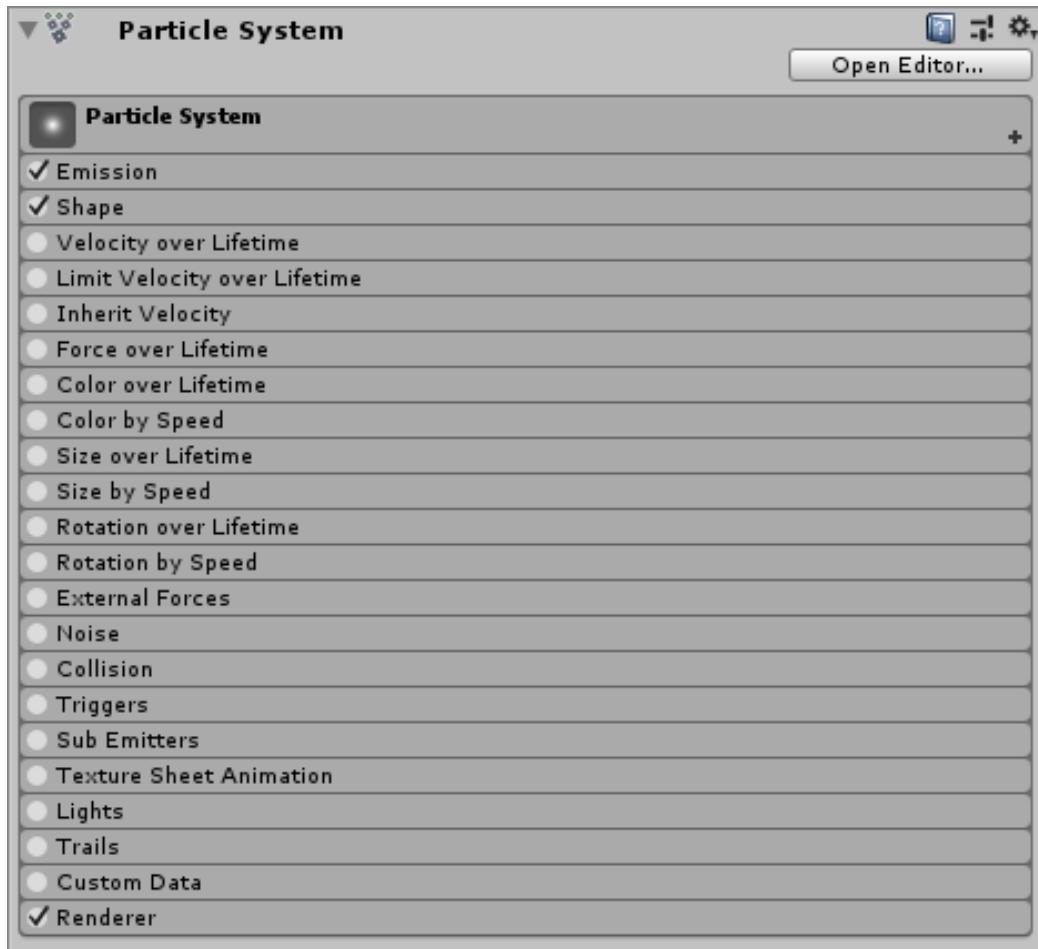


Figura 2.11: Questi sono i moduli del Particle System Shuriken

La figura 2.11 mostra i diversi moduli del Particle System Shuriken, ogni modulo presenta diversi parametri per modificare al meglio il sistema di particelle.

2.1.3 A* Pathfinding Project

A* Pathfinding Project[3] è un tool a cura di *Aron Grandberg* che fa utilizzo del famoso algoritmo A* per il calcolo del percorso minimo. Ora introdurre-

remo l'algoritmo per poi passare a descrivere nel dettaglio il tool. **A*** è un algoritmo di ricerca del tipo *best-first* su grafi che individua un percorso da un nodo iniziale verso un nodo di destinazione. Ogni nodo ha un costo di entrata che rappresenta la distanza tra il nodo corrente e quello di inizio, e un valore che indica la distanza euristica dal nodo corrente a quello di destinazione, il costo del nodo è composto dalla somma dei due descritti precedentemente. L'algoritmo sceglie il nodo con costo totale minore, ricalcola il costo dei nodi vicini a quello scelto, continua così fino ad arrivare a destinazione.

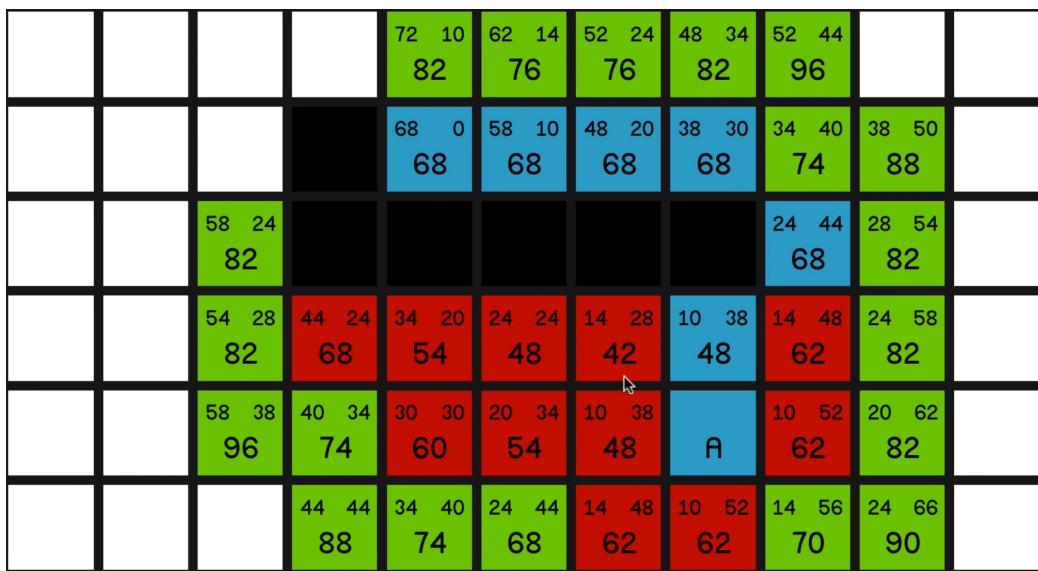


Figura 2.12: Questo è un esempio dell'algoritmo su un grafo a griglia

Nella figura 2.12 possiamo vedere il percorso minimo calcolato con l'algoritmo A* in blu, in rosso possiamo vedere i nodi controllati, in verde quelli non controllati, mentre in nero abbiamo i nodi definiti come "ostacolo".

Ora andiamo nel dettaglio del tool di Aron Granberg.

A pathfinding project* permette di scegliere tra diversi tipologie di grafi in modo tale da adattarlo ad ogni situazione. Le tipologie di grafi sono le seguenti:

- **Grafo a griglia**, come dice il nome è un grafo che genera nodi seguendo un pattern a griglia, è un pattern molto utile soprattutto quando bisogna aggiornare il grafo a runtime. Il grafo a griglia permette anche

di scegliere la forma effettiva dei nodi, come ad esempio è possibile scegliere tra nodi quadrati, esagonali o personalizzati; oltre ad ovviamente permettere di scegliere quanti vicini ha ogni nodo.

- **Grafo Navmesh**, è un grafo che esprime i dati sul pathfinding come mesh a forma di triangolo. È un grafo che permette il calcolo del percorso minimo molto velocemente anche perché contiene meno nodi del grafo a griglia, quindi l'algoritmo richiede meno ricerca.
- **Grafo a punti**, è il più semplice tra tutti i tipi di grafi, permette una elevata personalizzazione e consiste in un gruppo di punti collegati tra loro.

Una volta scelto il grafo è possibile scegliere anche un *layer* per indicare quali GameObject sono ostacoli in modo tale da evitarli. Per avere una personalizzazione completa del grafo è presente uno script, denominato *Graph Update Scene*, che permette di disegnare sul grafo delle aree e quindi impostare a piacimento i valori dei nodi di quell'area. Gli agenti che utilizzeranno il grafo per il calcolo del percorso minimo dovranno avere il due componenti principali, *AIPath*, il quale permette di impostare vari parametri, come la velocità dell'agente o l'accelerazione. *AIDestination Setter* il quale prende in input la destinazione dell'agente, il quale una volta impostata inizierà ad incamminarsi verso di essa.

2.1.4 EasyRoad3D e Bézier Path Creator

EasyRoad3D e Bézier Path Creator sono due tool che si basano sulle curve di Bézier, delle curve formate da 4 punti: uno di inizio, uno di fine e due per la curvatura.

EasyRoad3d permette, come si può intuire dal nome, creare facilmente delle strade su Unity. Il tool permette una grande personalizzazione delle strade, in più è tutto gestibile dall'inspector e dalla scena, infatti è possibile inserire dei punti per la creazione della strada con un semplice *shift+click*.

Bézier Path Creator permette di disegnare un percorso tramite un insieme di punti e di modificare la curvatura del percorso seguendo le regole delle curve di Bézier. Sul percorso è possibile far muovere un GameObject facilmente assegnandogli lo script *Path follower*, il quale permette anche di impostare la velocità.

2.1.5 Probuilder

Probuilder è un tool a cura di Unity stesso che permette di fare semplice modellazione direttamente in Unity, esso infatti è stato utilizzato recentemente per lo sviluppo di alcuni videogiochi, come ad esempio *SUPERHOT*, *Lucky's Tale*, *Tacoma*.

Oltre a fornire diversi strumenti per la modellazione 3D, permette anche di generare delle forme predefinite come scale, archi e porte. È inoltre fornito di un'interfaccia grafica che si sovrappone alla scena in modo tale da avere tutti i tool a portata di click.

2.1.6 Cinemachine

Cinemachine è un tool a cura di Unity che permette di gestire con molta facilità e senza bisogno di scrivere codice la telecamera in Unity, ad esempio è possibile disegnare un percorso da far seguire alla telecamera. Il tool si basa sulla gestione di telecamere virtuali gestite da un component definito *Cinemachine brain*, dal quale è possibile passare da una camera all'altra e gestire le diverse inquadrature. È possibile utilizzare anche una timeline per passare da una camera all'altra in un tempo specifico. Oltre ad essere stato utilizzato per la creazione di videogiochi, è stato utilizzato per la realizzazione di *Baymax Dreams*, un'animazione in collaborazione con *Disney Television Animation* con il quale hanno vinto un *Technology and Engineering Emmy Awards* il 7 aprile 2019.

Capitolo 3

Simulazione nel dettaglio

In questo capitolo si mostrerà il lavoro svolto nel dettaglio, per la realizzazione della simulazione. Partendo dalla simulazione base, per poi passare alla simulazione effettiva sulla strada di Obuse.

3.1 La Strada di prova

Come prima cosa è stata realizzata una semplice strada circolare con Easy-Road3D in modo tale da poter mettere gli oggetti di scena e iniziare a fare delle prove.

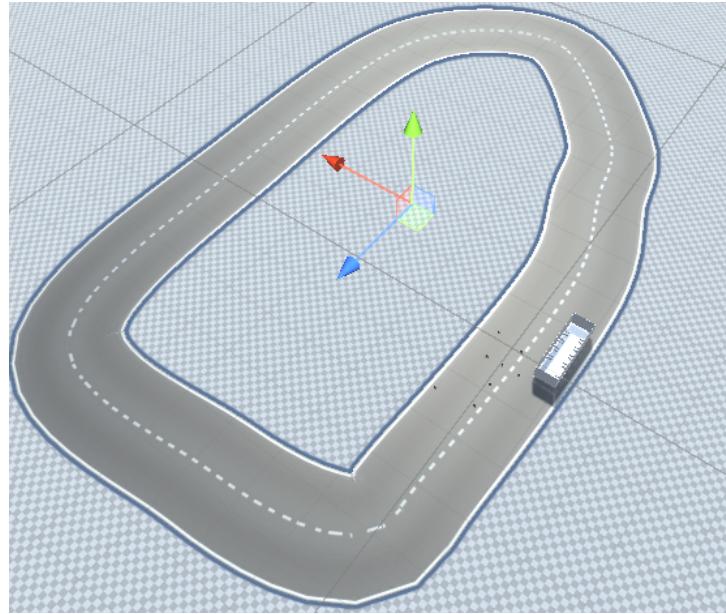


Figura 3.1: Questa è la strada di prova

Nella figura 3.1 mostra la strada di prova utilizzata l'inizio della realizzazione della simulazione.

Ora si parlerà dell'autobus, di come è stato creato ed utilizzato e di come ce ne siano state diverse versioni.



Figura 3.2: Questo è stato il primo bus della simulazione

3.2 Il mezzo di trasporto

Nella figura 3.2 possiamo vedere un scuolabus in *"low poly"* con tanto di animazioni e texture presenti nell'asset. Applicate delle piccole modifiche alle animazioni si è passati alla sperimentazione del tool A* pathfinding project, iniziando con un semplice cilindro come agente e provando a farlo salire sull'autobus. Dopo svariati tentativi nel quale non si era liberi di selezionare sezioni precise del bus dato il fatto che il modello scaricato era un blocco unico, compresi i posti a sedere e questa cosa era molto scomoda, soprattutto per gestire le sezioni calpestabili o meno del grafo di A*. Infatti non era permesso di selezionare le pareti dell'autobus per inserirle nel layer degli ostacoli in modo tale da indicarlo al grafo generato da A* Pathfinding Project.

Si è poi deciso di passare ad un bus molto più semplice, costituito da cubi base di unity.

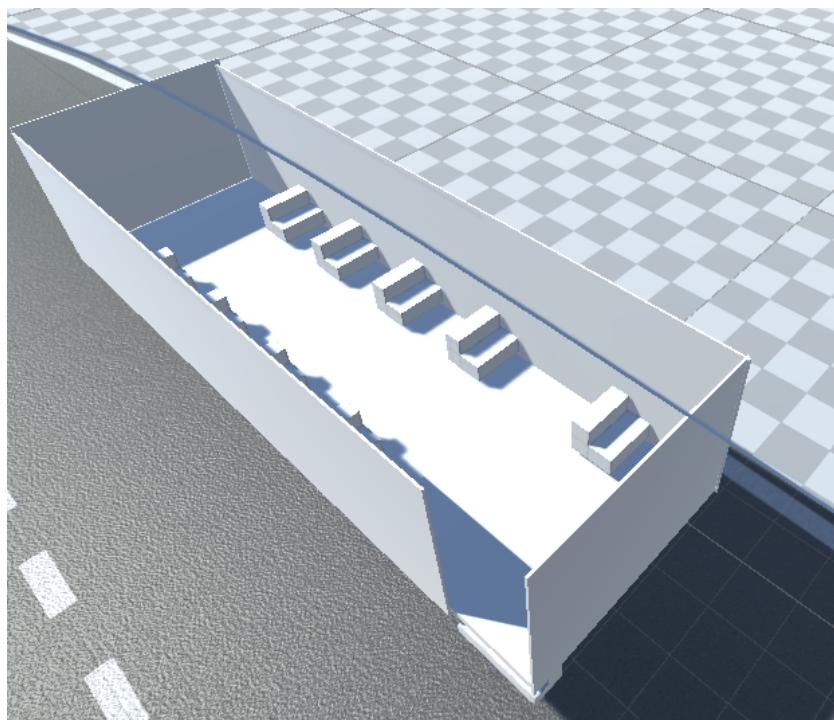


Figura 3.3: Questa è la prima versione dell'autobus realizzato in Unity

Si è poi deciso di utilizzare il tool di Unity *ProBuilder* per migliorare il bus della figura 3.3 aggiungendo finestre, ruote, la rampa e semplici texture.

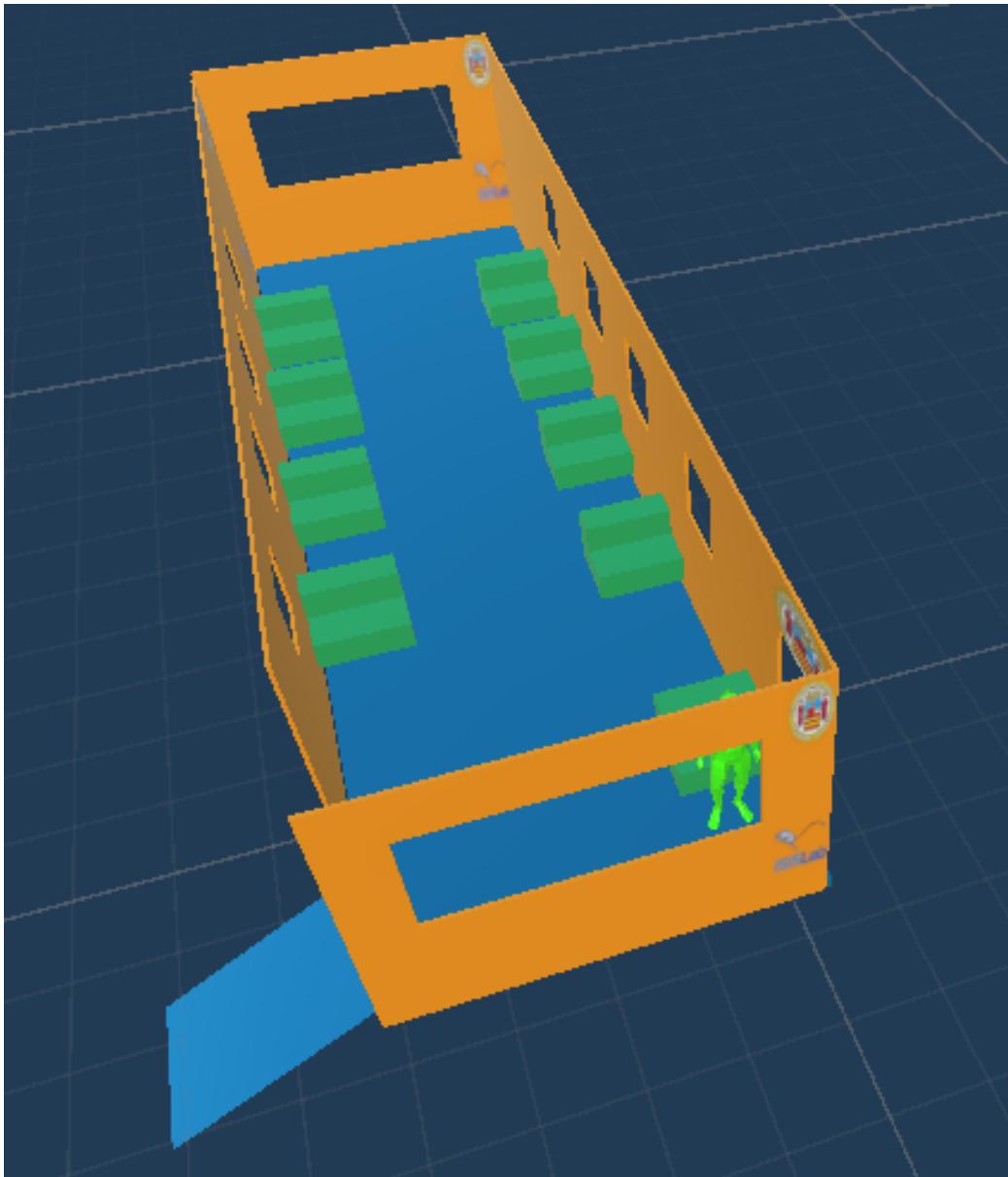


Figura 3.4: Questo è il bus finale ad 8 posti utilizzato per la simulazione di prova

Nella figura 3.4 si può vedere il bus creato con oggetti base di Unity per poi decidere di utilizzare il tool *ProBuilder*, il quale ha permesso di modellare

direttamente da unity senza l'utilizzo di applicativi esterni, per modificare al meglio le pareti in modo tale da creare le finestre per abbellire il modello. L'autobus presenta anche una rampa a scomparsa per rendere più facile la salita degli agenti sul bus. Con il nuovo autobus e la possibilità di assegnare i layer ai singoli elementi, è stato facilmente possibile far salire il primo agente sul bus.

Ora che la salita sul bus è facilmente applicabile si è passati ad aggiungere una lista di *"target"* davanti ai posti a sedere del bus, in modo tale da poterli assegnare ai vari agenti tramite script.

3.3 L'agente

Si è poi passati all'inserimento dei modelli degli agenti e quindi a lavorare sulle varie animazioni. Iniziando con l'animazione che va dallo stare in piedi allo stare seduto e poi quella da seduto allo stare in piedi.

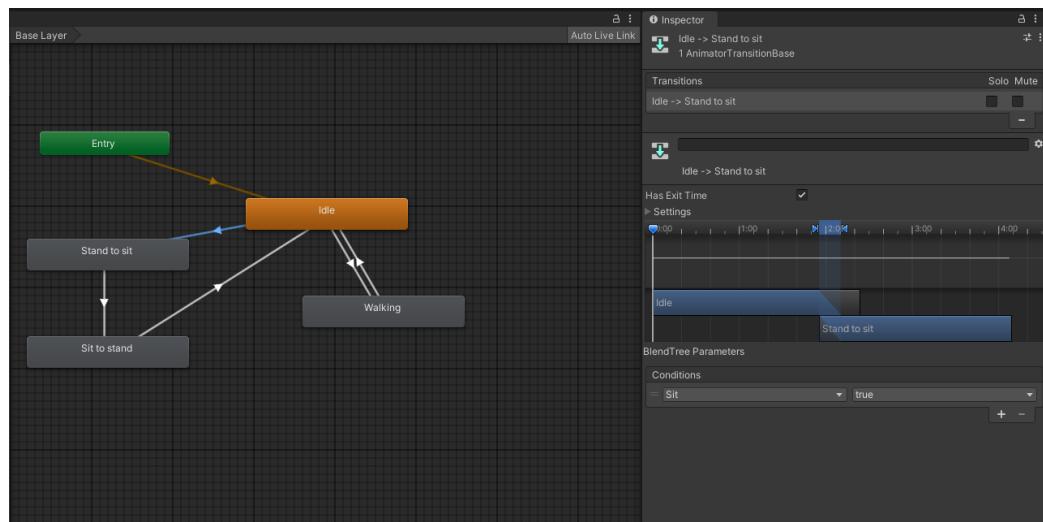


Figura 3.5: Questo è il component animator dell'agente

La figura 3.5 mostra sulla sinistra la macchina a stati finiti composta dagli stati in cui si può trovare l'agente, ogni stato ha la sua animazione. Sulla destra invece possiamo vedere una prospettiva sulla transizione che va dallo stato di *idle* a quello di *Stand to Sit*, come si può vedere in basso la

transizione avviene quando la variabile booleana Sit ha valore pari a true.

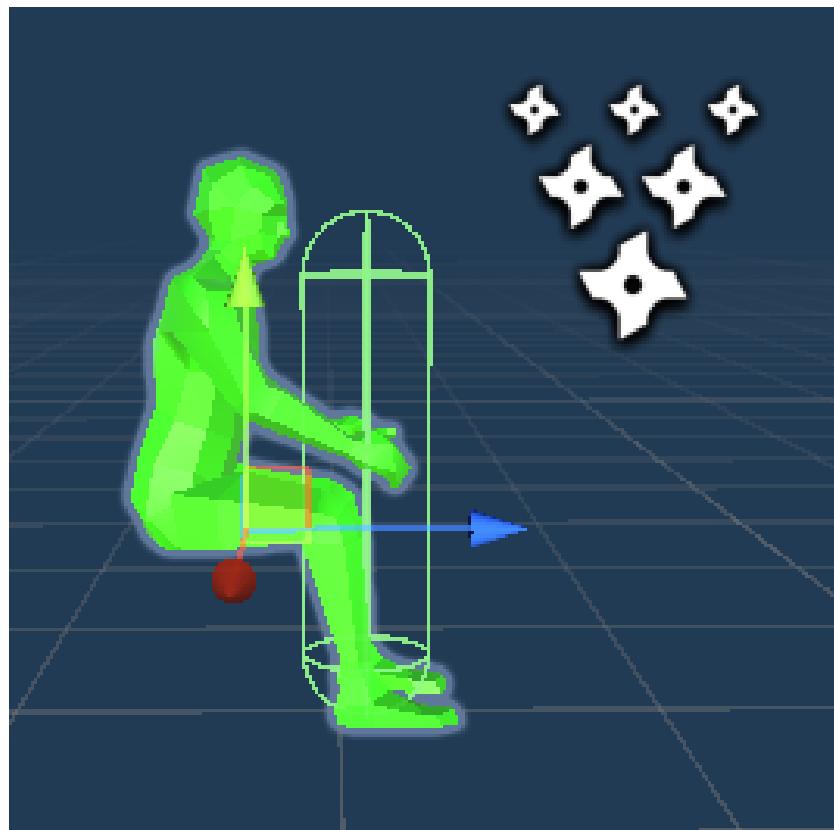


Figura 3.6: Frame dell'animazione di seduta

Nella figura 3.6 è mostrato un frame dell'animazione di seduta e il collider dell'agente che si restringe.

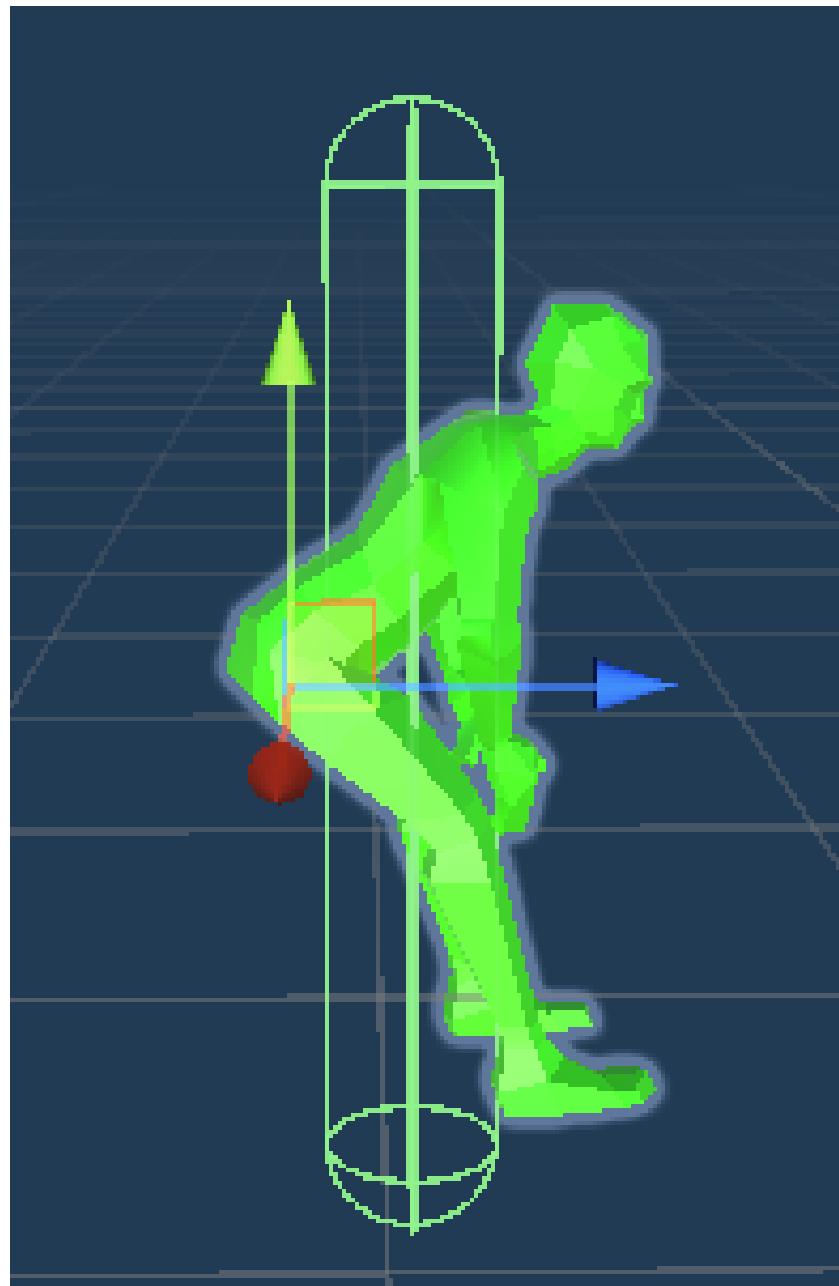


Figura 3.7: Frame dell'animazione di alzata

Nella figura 3.7 si può vedere un frame dell'animazione di alzata, denominata *Sit to Stand*, si può vedere anche il collider dell'agente che ritorna alla

forma originale.

Il passaggio delle varie animazioni è stato gestito da script assegnando valore alle variabili booleane dell'Animator.

```
1  public void TakeBus()
2  {
3      if (!bus.isMooving && bus.FreeTarget() != null &&
4          target== null)
5      {
6          animator.SetBool("Waiting", false);
7          target = bus.FreeTarget();
8          SetDestination(target);
9      }
10 }
```

Script 3.1: Metodo che permette agli agenti di prendere posto sull'autobus

Lo script 3.1 è il metodo all'interno della classe *Agent* con il quale l'agente prende un posto a sedere sul bus, rendendolo occupato ed inizia ad incamminarsi verso di esso. Come è possibile vedere dalla riga 5, si modifica la variabile booleana *Waiting*, all'interno del component *Animator* assegnandole valore pari a *false* così da passare dallo stato di *Waiting* allo stato di *Walking*.

A questo punto della simulazione gli agenti si sedevano correttamente, ma spesso capitava che facessero percorsi strani o che camminassero sopra i posti a sedere, per questo motivo al bus sono stati aggiunti diversi GameObject con component *Graph Update Scene*, in modo tale da poter penalizzare dei nodi del grafo per avere un percorso migliore per gli agenti.

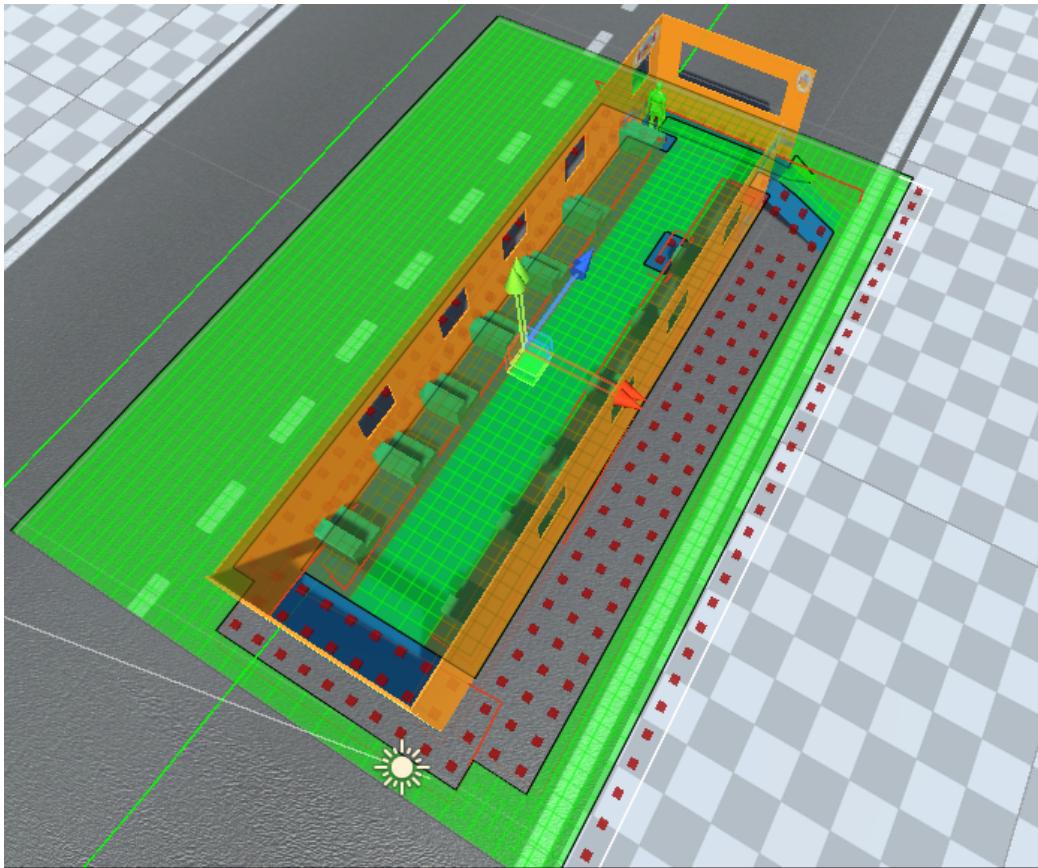


Figura 3.8: Questo è il grafo definitivo della simulazione.

Nella figura 3.8 possiamo vedere diverse zone nel grafo: le zone verdi sono aree completamente calpestabili, quindi non modificate dal *Graph Update Scene*; Le aree rosse sono aree calpestabili ma con penalità, per evitare che gli agenti scavalchino i sedili; le zone trasparenti con quadrati rossi sono zone non calpestabili. In più si può vedere il bus finale che è stato utilizzato nella simulazione definitiva di Obuse a 12 posti, in seguito se ne parlerà più approfonditamente. In seguito si è passati all'integrare il local avoidance tramite

Reciprocal Velocity Obstacles(RVO) del tool A* Pathfinding Project, il quale ha permesso di gestire facilmente il distanziamento tra gli agenti, impedendo penetrazioni e varie problematiche che potevano accadere implementando tale funzione. Un problema però è presente, alla salita degli agenti sul bus può capitare che distanziandosi sulla rampa, gli agenti cadano da essa finendo col camminare al di sotto del bus. Questo è stato risolto riposizionando, una volta arrivati a destinazione, gli agenti sul posto assegnato.

```

1  private void SitDown()
2  {
3      if (State == States.Healthy)
4      {
5          gameObject.AddComponent<ColliderCovid>();
6          GetComponent<ColliderCovid>().InfectionPercentage =
7          gameControl.infectionPercentage;
8      }
9      StartCoroutine(RotateOnSpot());
10     aiPath.enabled = false;
11     animator.SetBool("Waiting", true);
12     animator.SetBool("Sit", true);
13     rigidCube.tag = "Exiting";
14
15     transform.position = target.transform.position;
16     bool haveRigidbody = gameObject.GetComponent<Rigidbody>()
17     >() != null;
18
19     if (!haveRigidbody)
20     {
21         gameObject.AddComponent<Rigidbody>().freezeRotation =
22         true;
23     }
24 }
```

Script 3.2: Metodo per la seduta degli agenti

3.4 Il percorso del mezzo di trasporto

Come è possibile vedere dallo script 3.2 alla riga 14 viene risolto il problema dell'agente al di sotto del bus riposizionandolo nella giusta posizione.

Si è poi passati all'integrazione del tool *Beziér Path Creator*, per far muovere

l'autobus. Quindi si è tracciato un percorso sulla strada in modo tale che il bus lo seguisse.

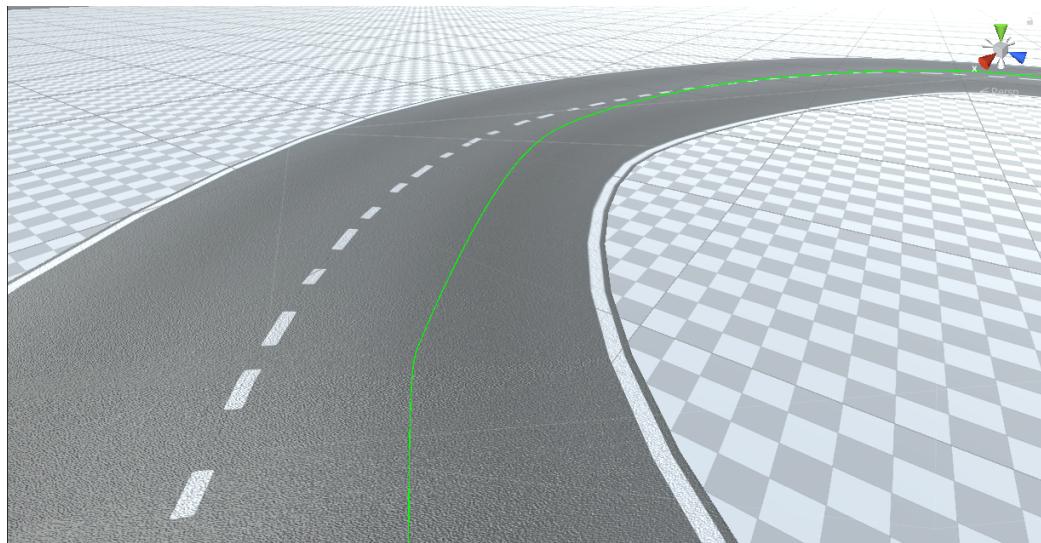


Figura 3.9: Questo è un tratto del percorso.

Nella figura 3.9 si vede, in verde, un tratto del percorso disegnato con Beziér Path Creator sul quale il bus si muove.

3.5 La fermata

Giunti alla situazione nella quale gli agenti salgono correttamente sul bus, prendono posto e si siedono, il bus si muove seguendo il percorso disegnato ci si è spostati sulla creazione di fermate per gli agenti, in modo tale da simulare le varie tratte dell'autobus, iniziando con una semplice struttura e una linea che funge da *trigger* per far sì che l'autobus si ferma.

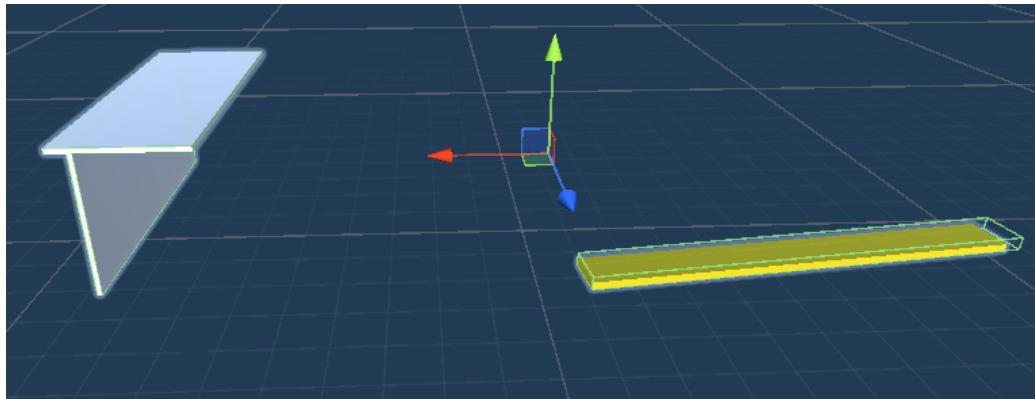


Figura 3.10: Fermata del bus

Nella figura 3.10 è possibile vedere una semplice struttura per dare l'idea di fermata, sulla sinistra, mentre sulla destra abbiamo la linea che fa da *trigger* per l'autobus.

È stato poi considerato utile aggiungere un GameObject denominato *Spawning Area*, all'interno di ciascuna fermata con il quale era possibile impostare la grandezza dell'area dove venivano generati gli agenti, deciderne il numero e decidere la velocità di ripartenza del bus. Questa scelta si è rivelata molto utile per passare dalla scena di prova a quella effettiva della simulazione.

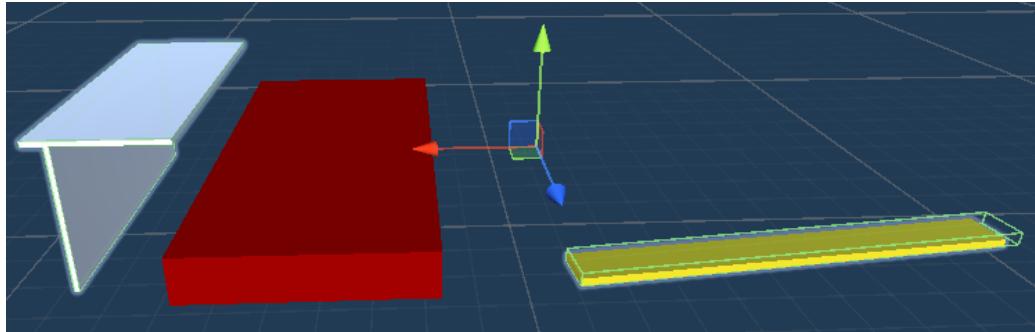


Figura 3.11: Fermata del bus con area di spawning

L'area rossa nella figura 3.11 è l'area di spawning, nella quale vengono generati gli agenti, di seguito nella figura 3.12 verrà mostrato nel dettaglio l'inspector dell'area.

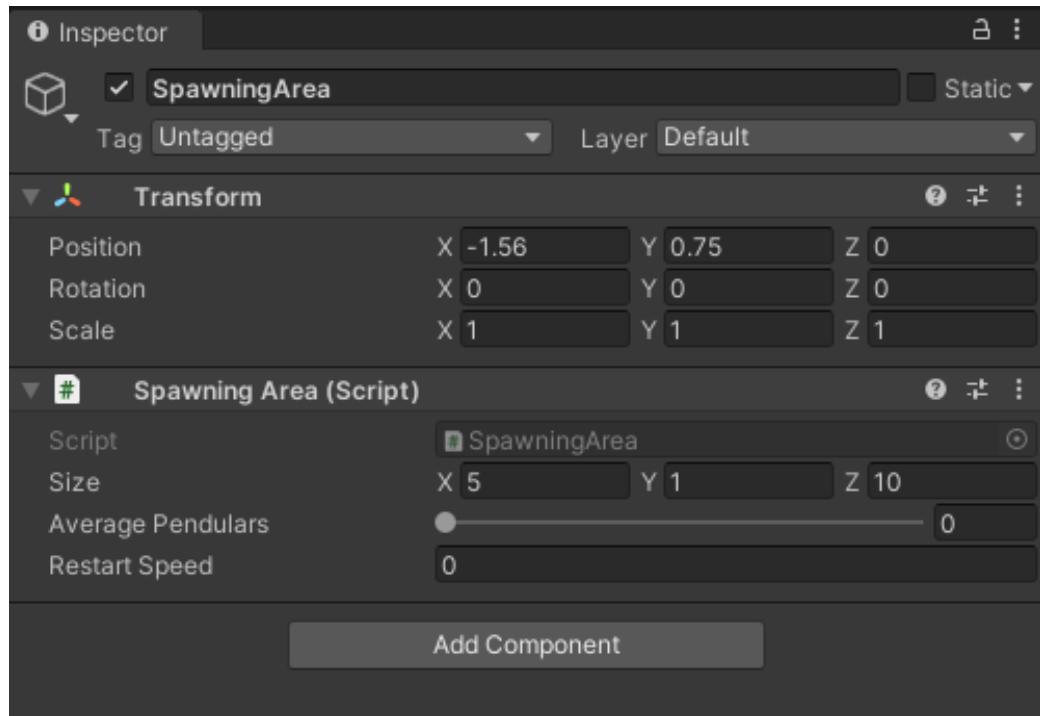


Figura 3.12: Inspector dell'area di spawning

Tramite la variabile *Size* è possibile modificare l'area nella quale verranno generati gli agenti; *Average Pendulars* permette di impostare il numero medio di agenti da generare alla fermata, il perché del numero medio di agenti invece di un numero fisso verrà spiegato in seguito; *Restart Speed* permette di decidere la velocità di ripartenza del bus in modo tale da poter far rispettare degli orari reali.

3.6 Il virus

La diffusione del virus è stata gestita tramite il particle system shuriken di Unity, il quale permette di gestire facilmente le collisioni con le particelle. Si è iniziati a realizzare un particle system che desse l'effetto dello spostamento d'aria, quindi un effetto di fumo che uscisse dalla bocca come è possibile vedere nella figura 3.13.

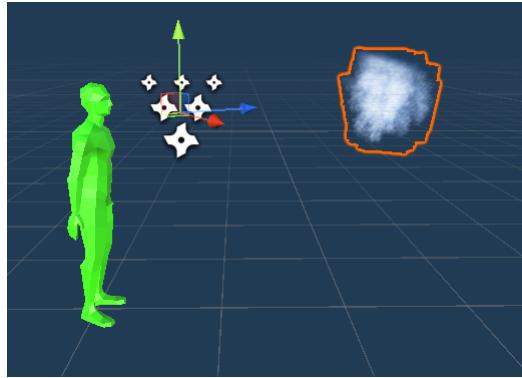


Figura 3.13: Effetto di fumo che parte dalla bocca dell’agente

Quindi si è passati alla realizzazione di un particle system che avesse un collider sulle particelle, un semplice effetto con delle sfere, inizialmente formate da un modello di CoVid-19 visibile nella figura 3.14

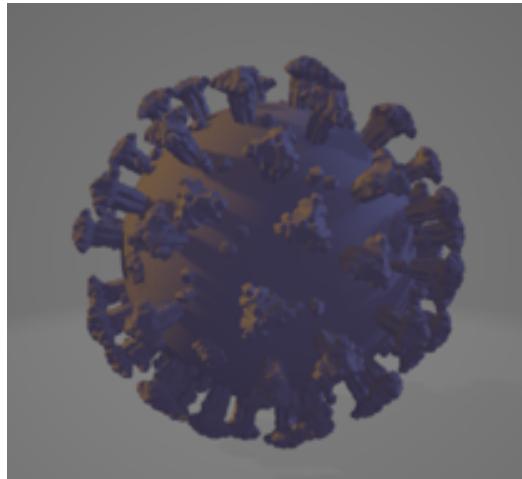


Figura 3.14: Modello dettagliato di CoViD-19

Per poi essere sostituito per ottimizzazione delle performance in seguito, ma questo se ne parlerà nella sezione dell’ottimizzazione.

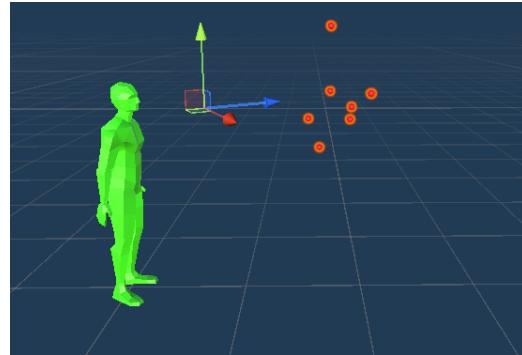


Figura 3.15: Effetto che simula le particelle infette di covid in un colpo di tosse

Nella figura 3.15 si vede l'effetto delle particelle che simulano quelle infette da covid, ogni particella ha un collider che permette di catturare le collisioni con il collider dell'agente sano che è possibile vedere nella figura 3.16

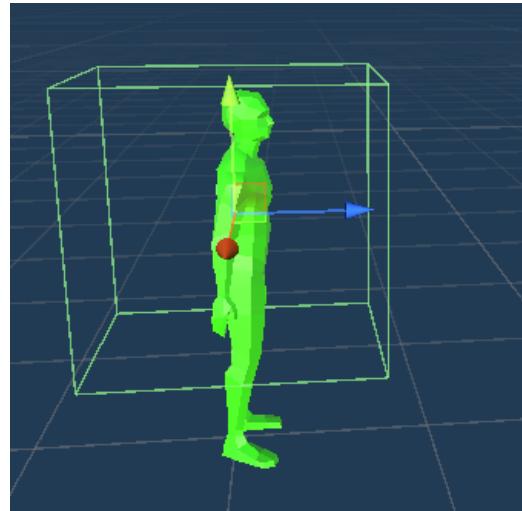


Figura 3.16: Il collider con il quale l'agente può essere infettato da una particella

Una volta che una particella di covid entra in collisione con il *collider* dell'agente sano, esso può diventare infetto, e quindi cambiare colore da verde a giallo

3.7 Il GameControl

Il GameControl è il GameObject che gestisce buona parte dei parametri della simulazione, infatti esso gestisce la generazione degli agenti, quanti agenti sono contagiosi all'interno della simulazione, la percentuale di che un agente venga infettato da una singola particella o il calcolare le statistiche della simulazione e scriverle su file.

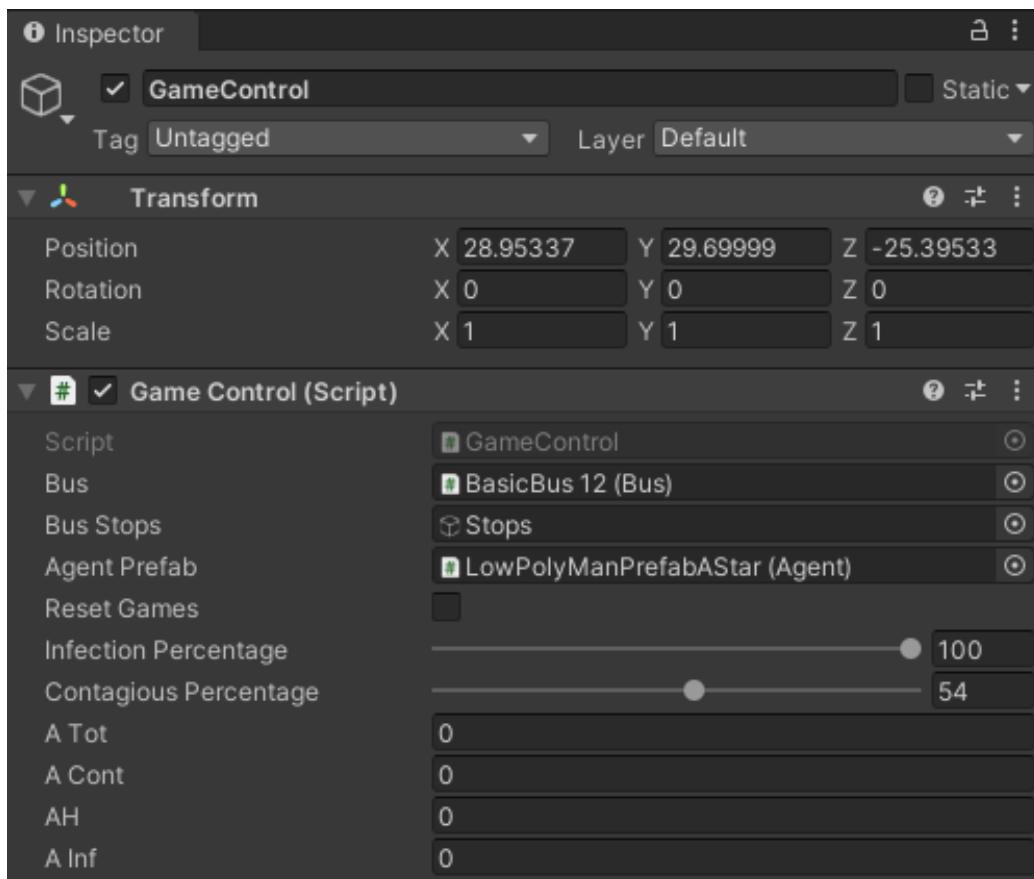


Figura 3.17: L'inspector del GameControl

Nella figura 3.17 è possibile vedere i parametri dell'oggetto GameControl citati precedentemente.

3.8 La tratta di Obuse

Con la simulazione funzionante si è poi passati alla ricerca di documenti su tratte di autobus reali, trovando un articolo riguardo al volume dei passeggeri sulla tratta di bus della cittadina giapponese Obuse [4], il volume è stato calcolato tramite una stima basata sui Wi-Fi scanner presenti sulla tratta.

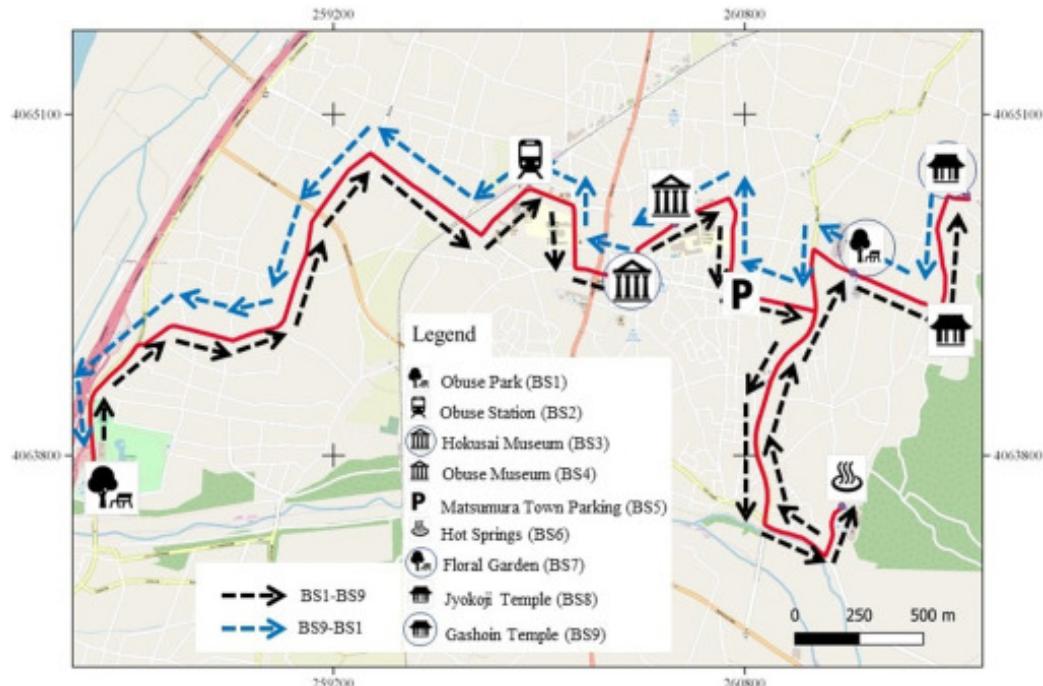


Figura 3.18: Mappa della tratta di Obuse

Nella figura 3.18 si vede, in rosso, la tratta del bus, mentre nella figura 3.19 si vede la media dei pendolari su ogni fermata della tratta.

Bus stop (BS)	Bus stop near tourism spot	Crowd average approximately (people)
BS1	Obuse park	10–15
BS2	Obuse station	15
BS3	Hokusai museum	5–7
BS4	Obuse museum	5–7
BS5	Matsumura town parking	5
BS6	Obuse hot spring	2–5
BS7	Floral garden	5
BS8	Jyokoji temple	2–5
BS9	Gashoin temple	2–5

Figura 3.19: Media di pendolari per ogni fermata della tratta

Considerando che i dati sono stati raccolti prima della pandemia da CoViD-19, è stato deciso di dimezzare i dati. Si è passati quindi dal bus a 24 posti ad uno a 12 posti, si è dimezzato il numero di pendolari per ogni fermata, e si è deciso di generare un numero di pendolari in un intervallo di $\pm 30\%$ della media calcolata tramite Wi-Fi scanner.

```

1  public void SpawningAtStops()
2  {
3
4      List<SpawningArea> areas = new List<SpawningArea>();
5      foreach (Transform stop in stops)
6      {
7          List<SpawningArea> spawnAreas=Utility<SpawningArea>.
8          GetAllChildren(stop.gameObject);
9          areas.Add(spawnAreas[0]);
10         areas.Add(spawnAreas[1]);
11     }
12     foreach (SpawningArea area in areas)
13     {
14

```

```

15     int effectivePendulars = Random.Range(area.
AvaragePendulars - area.AvaragePendulars * 30 / 100, area.
AvaragePendulars + area.AvaragePendulars * 30 / 100);
16
17     for (int i = 0; i < effectivePendulars; i++)
18         SpawnAgent(area);
19     }
20 }
21

```

Script 3.3: Genera i pendolari alle fermate

Nel codice 3.3, alla riga 15, è presente il codice che indica il range di $\pm 30\%$ della media.

Si è poi passati alla realizzazione in scala della strada su Unity utilizzando EasyRoad3D come è si vede nella figura 3.20, in verde è possibile anche vedere il percorso del bus realizzato con Bezièr Path Creator.

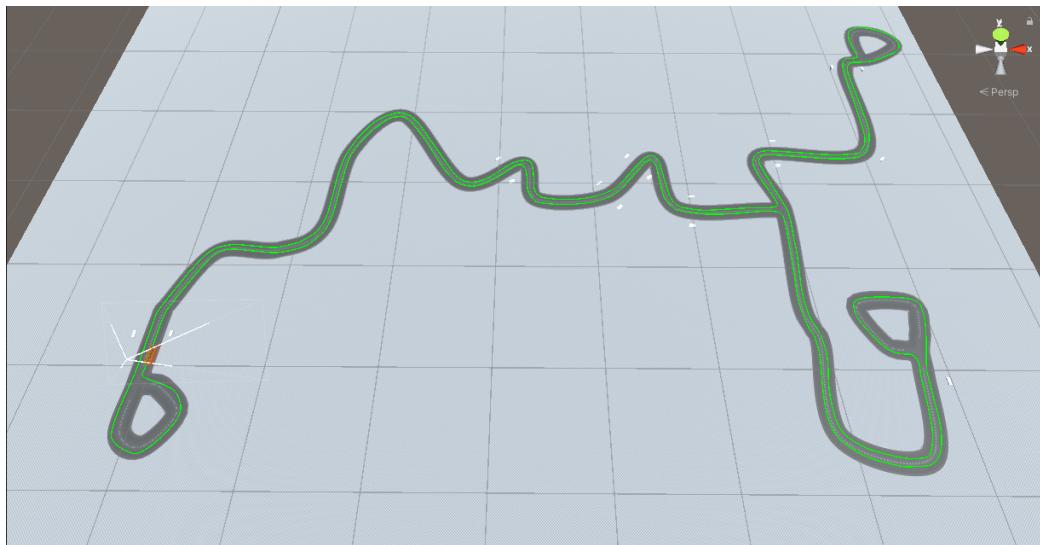


Figura 3.20: Strada di Obuse in Unity

3.9 L'interfaccia

Una volta arrivati alla simulazione funzionante, si è passati alla realizzazione di un'interfaccia grafica semplice ed intuitiva, inserendo l'orologio che scandisce il tempo reale, degli slider per la gestione dei parametri, dei pulsanti

per la gestione del tempo della simulazione e del testo per i dati sul contagio. L'interfaccia grafica si divide in due parti, quando la simulazione è in pausa e quando è in play.

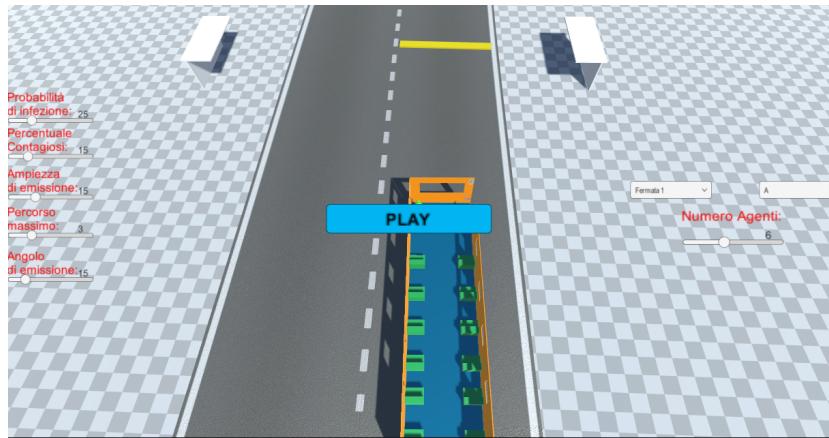


Figura 3.21: Interfaccia della simulazione in pausa

Nella figura 3.21 sono presenti diversi slider, sulla sinistra per la gestione dei parametri di contagio, sulla destra sono presenti due menu dropdown e uno slider per scegliere il numero di agenti da generare su una fermata selezionata dai menu, infine un pulsante al centro per far partire la simulazione.

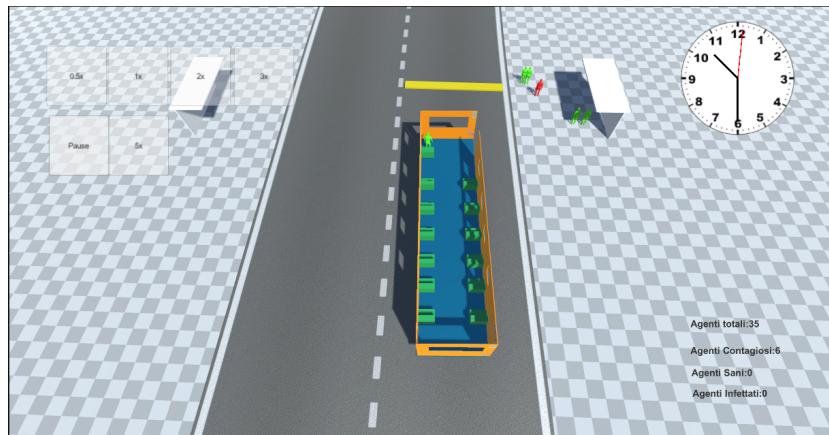


Figura 3.22: Interfaccia della simulazione in play

Nella figura 3.22 sono presenti, sulla sinistra i bottoni per la gestione del tempo, in alto a destra un orologio che scandisce il tempo reale della simulazione e in basso a destra sono presenti i dati del contagio della simulazione.

3.10 La telecamera

La telecamera è stata gestita completamente con il tool Cinemachine, come è possibile vedere nella figura 3.23, il quale ha permesso di creare una telecamera isometrica che seguisse il bus per l'intera simulazione senza scrivere codice.

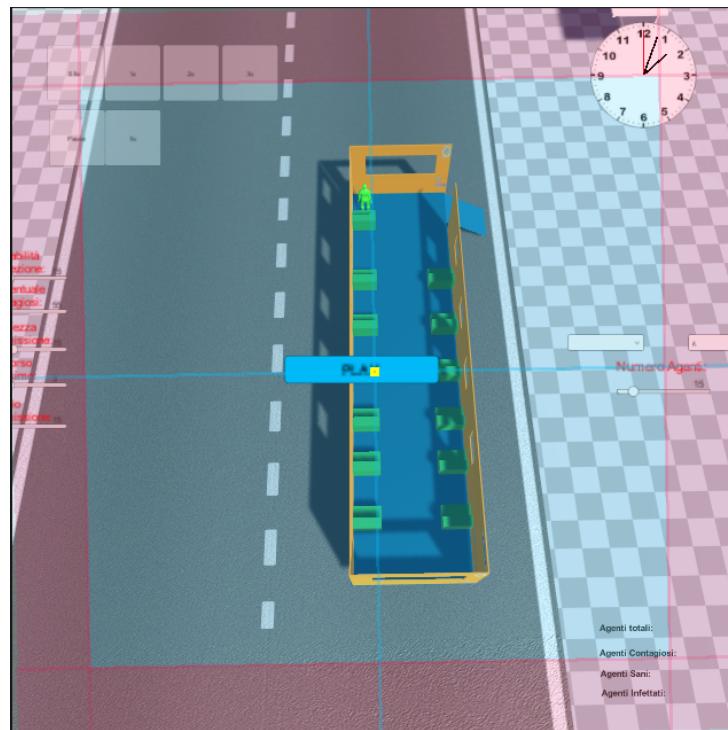


Figura 3.23: La griglia della telecamera, il puntino giallo indica l'oggetto da seguire

3.11 L'ottimizzazione

Il lavoro di ottimizzazione è iniziato nel momento in cui la simulazione aveva cali drastici di framerate, anche su diverse macchine. Tramite il *Profiler*, strumento presente in unity in grado di mostrare l'utilizzo di risorse della macchina e quale processo in particolare sta sfruttando le risorse, si è riusciti a comprendere quali erano i principali problemi. Come è possibile vedere dalla figura 3.24 ci sono dei cali drastici a 15 FPS che causano non pochi problemi.



Figura 3.24: Questo è il profiler della simulazione prima dei cambiamenti

Con il *profiler* alla mano e la simulazione in *play*, si è riusciti a comprendere in quali fasi avesse questi cali drastici. Come primo problema la telecamera era in una pessima angolazione, in alcune zone della tratta la posizione della telecamera le permetteva di inquadrare l'intera mappa, come è possibile vedere dalla figura 3.25, quindi renderizzando l'intero mondo della simulazione sfruttava molte risorse. Si è quindi ridotto la *distanza di render* della telecamera e si è passati ad un'inquadratura a *volo di uccello* facendo sì che non si veda la strada che viene generata.

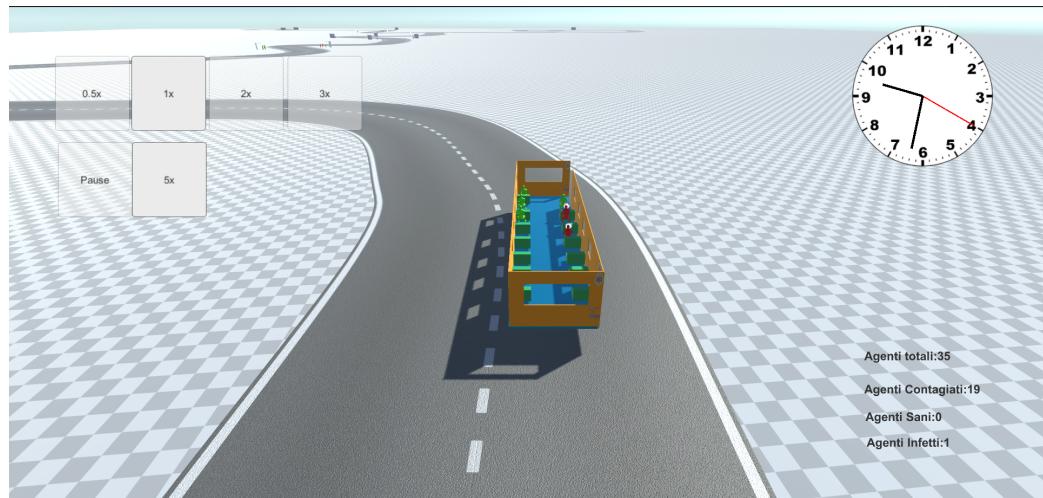


Figura 3.25: Questa è un frame nel quale la simulazione perdeva molti frame per secondo

Una volta risolto il problema della telecamera, si è notato che i cali erano, in frequenza minore, ma comunque presenti. Tramite il *profiler* ci si è accorti che il processo che sfruttava più risorse in certi momenti era il *particle system*, infatti i frame per secondo si abbassavano quando venivano emesse le particelle. Si è quindi passati a ridurre il numero di particelle emesse, da 10 ad 8, e si è cambiato il modello con il quale venivano generate, non più il modello nella figura 3.14 ma delle semplici sfere.



Figura 3.26: Questo è il profiler della simulazione a 5x

Ora la simulazione ha un'ottima media di frame per secondo, anche a 5x, come è possibile vedere nella figura 3.26.

3.12 La raccolta dei dati

Ad ogni avvio della simulazione viene generato un file contenente le informazioni riguardo il contagio, come è possibile vedere nella figura 3.27. Ad ogni tratta viene generata una riga contenente il numero di agenti totali, quelli contagiosi, quelli infetti e quelli sani.

1	2	3	4	5	6	7
1	Agenti Totali	Agenti Contagiosi	Agenti Infetti	Agenti Sani		
2	37	17	14	6		
3	34	19	11	4		
4	38	17	14	7		
5	34	19	10	5		
6	36	24	9	3		
7	39	21	15	3		
8	37	17	13	7		
9						
10						
11						
12						
13						

Figura 3.27: Raccolta dati di una simulazione completa

Bibliografia

- [1] Christine SM Currie, John W Fowler, Kathy Kotiadis, Thomas Monks, Bhakti Stephan Onggo, Duncan A Robertson, and Antuela A Tako. How simulation modelling can help reduce the impact of covid-19. *Journal of Simulation*, 14(2):83–97, 2020.
- [2] Navid Ghaffarzadegan and Hazhir Rahmandad. Simulation-based estimation of the spread of covid-19 in iran. *medRxiv*, 2020.
- [3] Aron Granberg. A* pathfinding project, 2011.
- [4] Arief Hidayat, Shintaro Terabe, and Hideki Yaginuma. Estimating bus passenger volume based on a wi-fi scanner survey. *Transportation Research Interdisciplinary Perspectives*, 6:100142, 2020.
- [5] Baichuan Mo, Kairui Feng, Yu Shen, Clarence Tam, Daqing Li, Yafeng Yin, and Jinhua Zhao. Modeling epidemic spreading through public transit using time-varying encounter network. *Transportation Research Part C: Emerging Technologies*, 122:102893, 2021.
- [6] David E Singh, Maria-Cristina Marinescu, Miguel Guzmán-Merino, Christian Durán, Concepción Delgado-Sanz, Diana Gomez-Barroso, and Jesus Carretero. Simulation of covid-19 propagation scenarios in the madrid metropolitan area. *Frontiers in public health*, 9:172, 2021.
- [7] Unity Technologies. Unity, 2005.
- [8] Nan Zhang, Wei Jia, Peihua Wang, Chung Hin Dung, Pengcheng Zhao, Kathy Leung, Boni Su, Reynold Cheng, and Yuguo Li. Changes in local travel behaviour before and during the covid-19 pandemic in hong kong. *Cities*, 112:103139, 02 2021.