

Università degli Studi di Salerno



FACOLTÀ DI INFORMATICA

Tesi di laurea  
in  
INFORMATICA

SIMULAZIONE DI ASSISTENZA AL  
PARCHEGGIO TRAMITE DEEP LEARNING

**Relatore:**

*Ch.mo. Prof. Andrea F. Abate*

**Correlatore:**

Dott. Ignazio Passero

**Candidato:**

*Francesco Abate  
Matr.05121/05354*

ANNO ACCADEMICO 2019/2020

# INDICE

1	Introduzione.....	8
1.1	Motivazioni.....	8
1.2	Struttura del documento.....	9
2	Fondamenti .....	10
2.1	ADAS .....	10
2.1.1	Autopilot e parcheggio assistito.....	11
2.2	Machine Learning .....	12
2.2.1	Neural Network .....	13
2.2.2	Deep Reinforcement Learning.....	14
2.3	Game Engine .....	15
3	Tecnologie e Strumenti .....	19
3.1	Unity 3D .....	19
3.1.1	Scripting e linguaggio C# .....	22
3.2	ML-Agents.....	22
3.2.1	Agente.....	23
3.2.2	Linguaggio Python.....	26
3.3	TensorFlow e TensorBoard .....	27
4	Implementazione .....	28
4.1	Scenario in Unity .....	29
4.2	Modello di automobile virtuale .....	33
4.2.1	Sottosistemi di semplificazione della guida .....	34
4.2.2	Automobile agente e sensori di guida.....	36
4.3	Educare l'agente .....	42
4.3.1	Sistema di ricompense .....	43
4.3.2	Sottosistema di ricompensa distanziale .....	46
4.3.3	Sottosistema di ricompensa direzionale.....	47
5	Apprendimento .....	56
5.1	Introduzione alle sessioni FSk e LSk.....	57
5.2	Struttura delle sessioni .....	57
5.3	Sessioni FSk.....	58
5.3.1	Parking-FS1-X .....	59
5.3.2	Parking-FS2-X .....	60
5.3.3	Parking-FS3-X .....	62

5.3.4	Parking-FS4-X.....	63
5.3	Sessioni LSk .....	64
5.4.1	Parking-LS1-X.....	65
5.4.2	Parking-LS2-X.....	69
5.4.3	Parking-LS3-X.....	74
6	Prototipo e sviluppi futuri.....	77
6.1	Build .....	77
6.1.1	Funzionamento del menu.....	78
6.2	Sviluppi futuri.....	80
	Bibliografia.....	83

A chi da una vita ha sempre creduto in me.  
E che continuerà sempre a farlo.

## **Abstract**

La guida autonoma è una delle sfide più interessanti a cui la ricerca si dedica: lo scopo è creare un sistema di guida autonoma il più affidabile e sicuro possibile, al fine di tutelare l'incolumità del conducente alla guida, degli altri veicoli e dei pedoni. Negli ultimi anni, le auto sono state dotate di sistemi elettronici di assistenza alla guida chiamati Advanced Driver Assistance Systems (ADAS), con i quali si intendono sistemi avanzati elettronici che aiutano il conducente nell'adottare una condotta di guida sicura per tutti.

Questa tesi utilizza il gaming engine Unity 3D per simulare l'applicazione di un sistema di guida automatica, realizzando i sensori di parcheggio e di riconoscimento automatico di ostacoli, installati su un'auto posta in un ambiente 3D modellato come un tipico contesto urbano. L'auto, muovendosi nell'ambiente, addestrerà il sistema di guida automatica tramite tecniche di Deep Reinforcement Learning: imparerà ad effettuare manovre e a rilevare il parcheggio e gli ostacoli tramite due appositi sensori modellati a bordo vettura. Il Deep Reinforcement Learning è supportato dal progetto open-source ML-Agents, il quale permette il training di agenti intelligenti tramite apposite API di Python in ambienti virtuali 2D/3D.

Lo scopo della tesi è sperimentare il funzionamento dei sistemi ADAS elencati e dimostrare quanto sia conveniente e sicuro usufruirne: come caso di studio, verrà mostrata l'applicazione in un parcheggio composto da molteplici posti auto, dei quali solo uno sarà libero. Per effetto dell'addestramento, l'auto si dirigerà verso il posto libero evitando incidenti con le auto parcheggiate tramite il sistema di guida automatica e il sensore di rilevamento ostacoli, per poi effettuare manovre ed entrare correttamente nel posto dedicato. Come mostrato dal caso di studio, l'esperienza di tesi, ha permesso di realizzare un utile banco di prova per la simulazione di algoritmi di assistenza alla guida basati su Machine Learning.

# Ringraziamenti

Sono molte le persone che hanno contribuito alla mia crescita personale e al diventare ciò che oggi sono. Alcune di queste, meritano un ringraziamento speciale.

In primis ringrazio i miei genitori per tutto il supporto che mi hanno sempre dato, i quali sono sempre stati, e sempre saranno, un grande esempio da seguire.

Un ringraziamento speciale alla mia ragazza, Valentina, luce dei miei occhi, per avermi sempre sostenuto, per la tenacia e la grinta di cui mi ha caratterizzato durante gli esami e per l'incoraggiamento che mi fornisce ogni giorno in ciò che faccio.

Ringrazio il mio migliore amico, Daniele, perché avere un fratello diverso nel mondo con il quale confidarmi e condividere qualsiasi ricordo è un qualcosa di unico e bellissimo.

Ringrazio il mio gruppo di amici, Davide, Giuseppe, Francesco, Luigi e Lorenzo per il supporto costante e per i momenti di distrazione che decoreranno sempre la mia vita.

Ringrazio il mio gruppo universitario e i colleghi con i quali ho sostenuto i progetti durante il percorso di laurea. Un grazie grande quanto il bene che gli voglio in particolare a Giannandrea, Carmine, Rosario, Marco, Vincenzo, Emmanuel e Simone per tutti gli aiuti che mi hanno dato e perché hanno reso i miei anni di università molto più leggeri e pieni di divertimento.

Un carissimo ringraziamento alla Kineton, al professore Andrea F. Abate e al dottore Ignazio Passero per aver suscitato in me l'amore verso la computer graphic e lo sviluppo videoludico, per la magnifica opportunità di tirocinio e per la pazienza prestatami nella stesura della tesi. Un grande grazie in particolare al dottore Ignazio Passero per tutti i consigli, di cui farò tesoro, che mi ha dato durante l'esperienza di tirocinio. Inoltre, gli devo gran parte dell'amore che ho verso il mondo dell'intelligenza artificiale.

Ringrazio il professore Deufemia, che con il primo 30 mi ha spinto a credere molto di più in me stesso e a dare sempre di più, fino al passare i miei stessi limiti.

Ringrazio i professori Rescigno e De Lucia, dai quali ho appreso quanto sia importante la precisione e la rigorosità in ciò che si fa.

Ringrazio tantissimo il professore Carrabs per la passione in me suscitata verso la matematica, la quale vedeva con occhi diversi: è stato l'unico professore a farmi approcciare ad una materia tanto ostile con una semplicità estrema.

La persona che sono oggi non sarebbe esistita senza di loro.



# Capitolo 1

## Introduzione

### 1.1 Motivazioni

Il continuo progresso tecnologico ha portato i sistemi di trasporto ad essere soggetti a grandi trasformazioni, partendo dall'essere semplici entità meccaniche all'essere piattaforme intelligenti interconnesse: ciò è stato reso possibile dai sistemi ADAS, i quali aiutano il conducente nel processo di guida.

Si prospetta che la standardizzazione di tali sistemi migliori notevolmente la qualità della vita, riducendo quanto possibile gli incidenti provocati da distrazioni da parte dei conducenti.

Lo scopo dei sistemi ADAS consiste nell'assistere il conducente autonomo con un software intelligente, il quale farà uso di molteplici sensori al fine di condurre una guida quanto più sicura possibile.

Questa tesi tratta la casistica di parcheggio automatizzato, quindi l'auto farà uso del sistema di guida automatica e di un sensore per il rilevamento degli ostacoli al fine di effettuare un parcheggio corretto.

Il sistema di guida automatica si approccia al Deep Reinforcement Learning, il quale consiste nell'imparare e migliorare tramite molteplici tentativi ed errori durante sessioni di addestramento.

Tale tesi nasce da un'innata passione verso il Machine Learning, dal voler scoprire come un oggetto relativamente stupido possa effettuare scelte intelligenti. La tipologia di apprendimento utilizzata si è dimostrata estremamente interessante, siccome è insolito osservare un'auto che impara tramite sensori a guidare autonomamente e ad evitare collisioni con ostacoli ed altri veicoli.

## 1.2 Struttura del documento

La tesi è articolata come segue:

- Nel primo capitolo si presenta un'introduzione dell'argomento trattato, definendo il problema e ciò che mi ha spinto a stendere questa tesi;
- Il secondo capitolo offre nozioni teoriche riguardanti i concetti su cui la tesi si basa;
- Il terzo capitolo illustra gli strumenti utilizzati per la realizzazione del progetto;
- Il quarto capitolo riguarda la struttura dell'ambiente simulato e l'implementazione dell'auto e dei sensori;
- Il quinto capitolo parla delle sessioni di test svolte e dei risultati ottenuti.

# Capitolo 2

## Fondamenti

### 2.1 ADAS

Ogni anno circa 1.35 milioni di persone muoiono in un incidente stradale; tra 20 e 50 milioni di persone rimangono gravemente ferite, spesso riducendosi ad una disabilità perenne. Le cause sono molteplici, vanno dalla distrazione umana all'uso dell'alcol e sostanze stupefacenti, dalle strade che non sono delle migliori ai veicoli poco sicuri [1]. La tecnologia ha sempre mirato a migliorare la sicurezza alla guida, fornendo alle auto sistemi di avvertimento fino ad arrivare a sistemi di guida automatici. Un esempio palese è l'auto Tesla, la quale presenta un hardware avanzato in grado di offrire le funzioni di autopilot tramite sensori e videocamere, le quali rilevano oggetti nell'ambiente circostante [2]. Tali sistemi si presentano sotto il nome di Advanced Driver Assistance Systems (ADAS) e sono tutt'ora oggetto di ricerche e di sviluppo. Tali sistemi sono installati sulle vetture di nuova omologazione e si prospetta che nei prossimi anni diventino uno standard. Alcuni di questi sono il parcheggio assistito, il

controllo adattivo della velocità, l'anticollisione e il sistema di rilevamento pedoni: ciò avviene tramite il riconoscimento tridimensionale dell'ambiente circostante.

Ovviamente, tali sistemi servono a migliorare la sicurezza di guida con l'obiettivo di ridurre al minimo i rischi di incidente ed agevolare la guida di un'automobile [3].

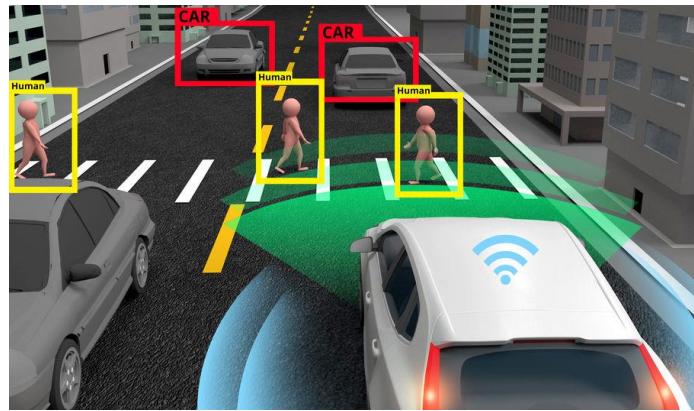


Figura 1 - Riconoscimento con sistemi ADAS

### 2.1.1 Autopilot e parcheggio assistito

Un'auto con funzioni di autopilot è capace di percepire l'ambiente circostante senza l'aiuto del conducente e di raggiungere la destinazione in maniera totalmente autonoma. Il veicolo conserva una mappa dei percorsi percorribili, in modo da poter trovare il percorso ottimale da percorrere per arrivare alla destinazione: fatto ciò, entrano in gioco sensori e videocamere, i quali vengono utilizzati per scansionare l'ambiente circostante e rielaborarlo in un'immagine tridimensionale. Dalle scansioni continue dei sensori sarà possibile rilevare ostacoli e misurare distanze, quindi elaborare scelte intelligenti in base a ciò che circonda il veicolo. Inoltre, alcuni sensori possono individuare la distanza tra il veicolo e l'ostacolo, in modo da avere maggiore precisione nelle manovre [4].



Figura 2 - Parcheggio assistito tramite sensori

## 2.2 Machine Learning

Machine Learning è la materia che insegna ai dispositivi come fare qualcosa che ad umani e animali verrebbe naturale [5]: il dispositivo impara a prendere decisioni e predizioni tramite apposite tipologie di apprendimento e migliora le proprie scelte nel tempo tramite tanta esperienza. Alla base dell'apprendimento automatico ci sono una serie di algoritmi che permetteranno di prendere specifiche decisioni e di riutilizzare azioni apprese nel tempo. È un ramo notevolmente vasto, siccome prevede differenti modalità e tecniche che variano in base alla situazione da affrontare [6].

L'automobilistica è uno dei tanti campi influenzati dal Machine Learning, il quale viene utilizzato anche nella finanza, nella computer grafica, in biologia, nella produzione di energia, eccetera [5]. Spesso si pensa che il Machine Learning si utilizzi in campi specifici o in determinati settori di ricerca, quando poi se ne fa un uso quotidiano senza accorgersene: un'applicazione classica è quella del riconoscimento vocale di cui sono dotati molti smartphone, il sistema delle pubblicità traccianti, cioè la visualizzazione delle pubblicità in base ai propri interessi e così via.

Il Machine Learning è un campo tuttora in evoluzione, ramificato in tanti altri campi non obbligatoriamente correlati all'informatica. Piuttosto, l'unico fattore limitante potrebbe essere il timore dell'uomo riguardante il fatto che le macchine possano divenire troppo intelligenti [6].



*Figura 3 - "La gente ha paura del fatto che i computer diventino troppo intelligenti e dominino il mondo, ma il vero problema è che pur essendo ancora troppo stupidi lo hanno già conquistato" - cit. Prof. Pedro Domingos*

### 2.2.1 Neural Network

Le Neural Network sono un sistema di apprendimento in cui viene utilizzata una rete di funzioni per comprendere ed elaborare un input in un determinato output [7]; vengono utilizzate come set di algoritmi per il Machine Learning in molteplici ambiti.

Il concetto di Neural Network artificiale si approccia alla Neural Network biologica, la quale permette ad ogni individuo di ragionare, fare calcoli, riconoscere immagini e suoni, imparare e agire tramite i propri neuroni: l'idea è quella di poter replicare artificialmente il cervello umano su dispositivi hardware [8].

Le Neural Network artificiali sono anch'esse composte da neuroni, i quali prendono degli input, li elaborano e producono un output [9].

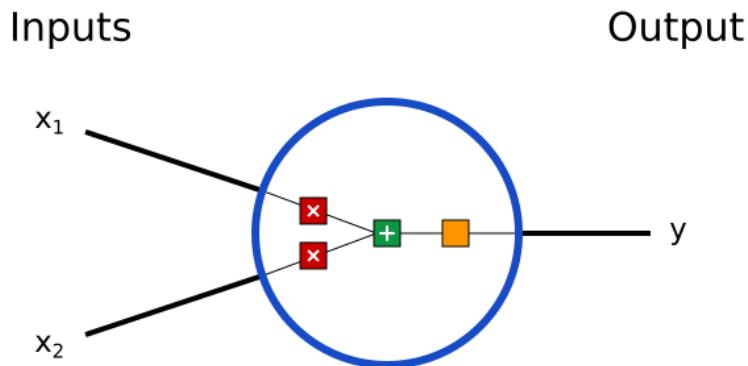


Figura 4 - Neurone di una Neural Network

Ogni neurone elabora determinati input in base a quanto essi siano rilevanti: se un input viene ritenuto più utile o conveniente, acquisirà più importanza nel calcolo dell'output [10].

Volendo fare un esempio reale, poniamo il caso in cui l'auto (che denominiamo agente) debba effettuare un parcheggio posto tra due auto statiche: se l'agente si sta muovendo e si trova troppo vicino all'auto posta in avanti, la retromarcia sarà la direzione preferibile da seguire, altrimenti si andrebbe incontro ad una collisione.

La Neural Network divide i propri neuroni in livelli [9], quali sono:

- Input Layer, nel quale i neuroni effettuano semplici operazioni basilari [10];
- Hidden Layer, livelli nei quali i neuroni effettuano decisioni in base ai risultati dati dai neuroni dei precedenti livelli, i quali potranno essere a loro volta altri neuroni appartenenti ad altri Hidden Layer [10];

- Output Layer, contiene un unico neurone che fornirà l'output della Neural Network [10].

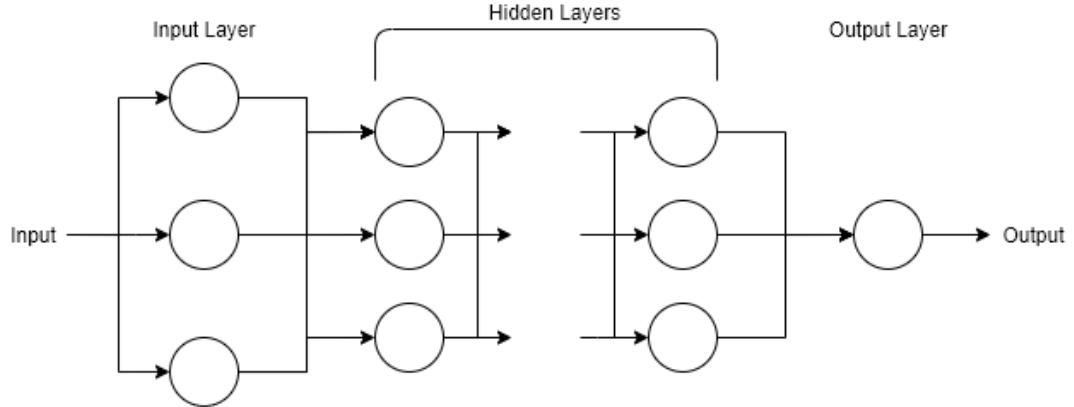


Figura 5 - Neural Network suddivisa in layer con N Hidden Layer

## 2.2.2 Deep Reinforcement Learning

Il Reinforcement Learning è un tipo di apprendimento della branca del Machine Learning: l'agente (colui che intraprende decisioni e impara) assume i criteri secondo cui prendere decisioni tramite un sistema di ricompense e penalità: in sostanza, l'agente impara come comportarsi ricevendo costantemente feedback in conseguenza dell'azione intrapresa, che siano positivi o negativi.

L'agente affronterà una lunga serie di test al fine di provare molteplici situazioni secondo diversi input, in modo tale da comprendere cosa è giusto e cosa è sbagliato tramite feedback: comprenderà la soluzione migliore percorrendo tutte le azioni che hanno portato a feedback positivi, in modo da ricevere la ricompensa più alta possibile [11].

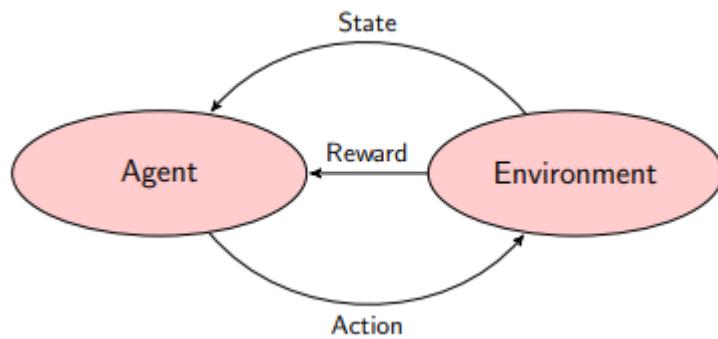


Figura 6 - Modello Reinforcement Learning

Il Deep Learning è un altro tipo di apprendimento che consiste nel riconoscimento di dati a partire dall'utilizzo di dati già esistenti. Un esempio palese di tale tecnica è il

riconoscimento facciale di una persona: verrà riconosciuta grazie a diverse caratteristiche che la contraddistinguono da altre persone.

Reinforcement Learning e Deep Learning sono due tipologie di apprendimento non mutualmente esclusive: difatti, si parla di Deep Reinforcement Learning. In tal caso, l'agente impara ad eseguire azioni e ad intraprendere azioni tramite il sistema di ricompense e tramite il riconoscimento di dati [12].

Nel caso dell'automobile in un parcheggio, il Deep Reinforcement Learning consiste nel farle comprendere dove parcheggiare grazie al riconoscimento visuale del posto auto e nel farle comprendere come parcheggiare tramite il sistema di ricompense, il quale assegnerà reward negativi in caso di collisione con altre auto e reward positivi in caso di parcheggio corretto.

## 2.3 Game Engine

I Game Engine sono softwares molto sofisticati in grado di offrire ai Game Creator determinate funzioni utili alla creazione di videogiochi in maniera relativamente rapida ed efficiente: permettono, in genere, di importare materiali 2D e 3D, creare ambientazioni che simulino la realtà, aggiungere effetti audio e visivi, creare animazioni e aggiungere logica nel comportamento degli oggetti [13]. I Game Engine riducono il costo, la complessità e il time-to-market che si riscontrerebbero nella creazione di un videogioco senza un Game Engine. Sono estremamente efficienti e riducono addirittura le conoscenze necessarie per la creazione di un videogioco [15].

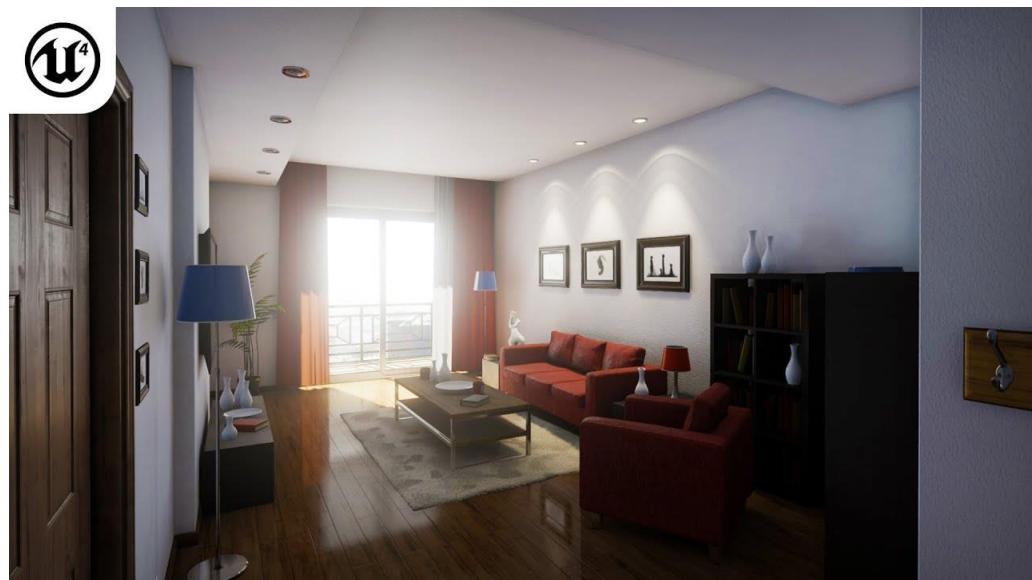


Figura 7 - Ambientazione 3D in Unreal Engine 4

È possibile vedere il Game Engine come un programma che raggruppi molteplici sottosistemi, detti anche manager, ognuno dei quali copre un preciso aspetto di un qualunque videogioco.

Il Render Manager permette il caricamento e l'elaborazione di file grafici per poi mostrare a video ciò che il giocatore visualizza tramite hardware grafico e l'aiuto di librerie grafiche come OpenGL e DirectX [14].

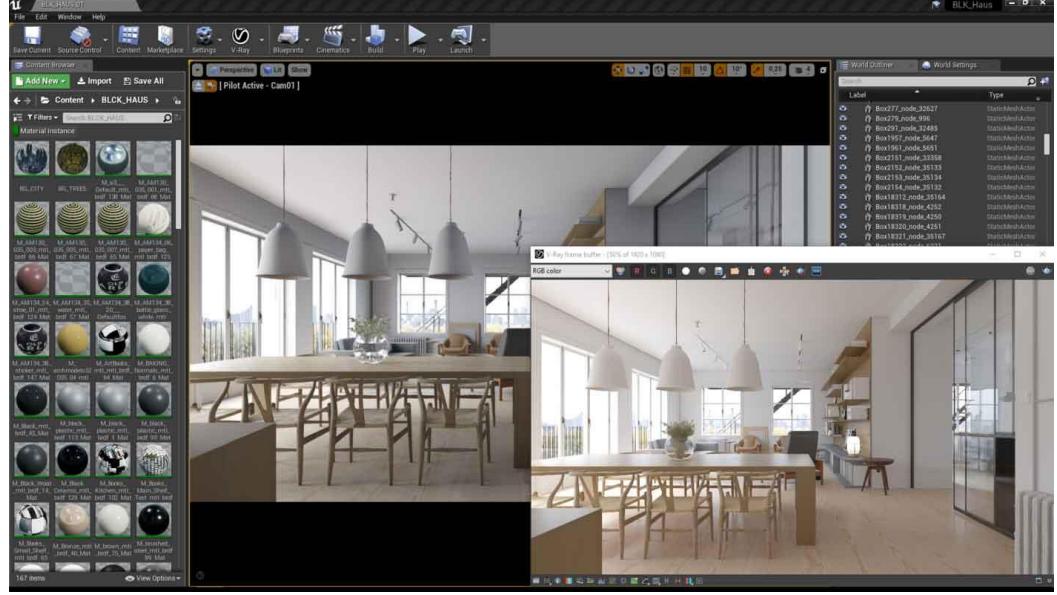


Figura 8 - Rendering in Unreal Engine 4

L'Input Manager permette la ricezione, tramite apposito hardware, di comandi dettati dall'utente al fine di compiere azioni nell'ambiente virtuale simulato. I dispositivi di input solitamente utilizzati dall'utente sono la tastiera, il mouse e il joypad [14].



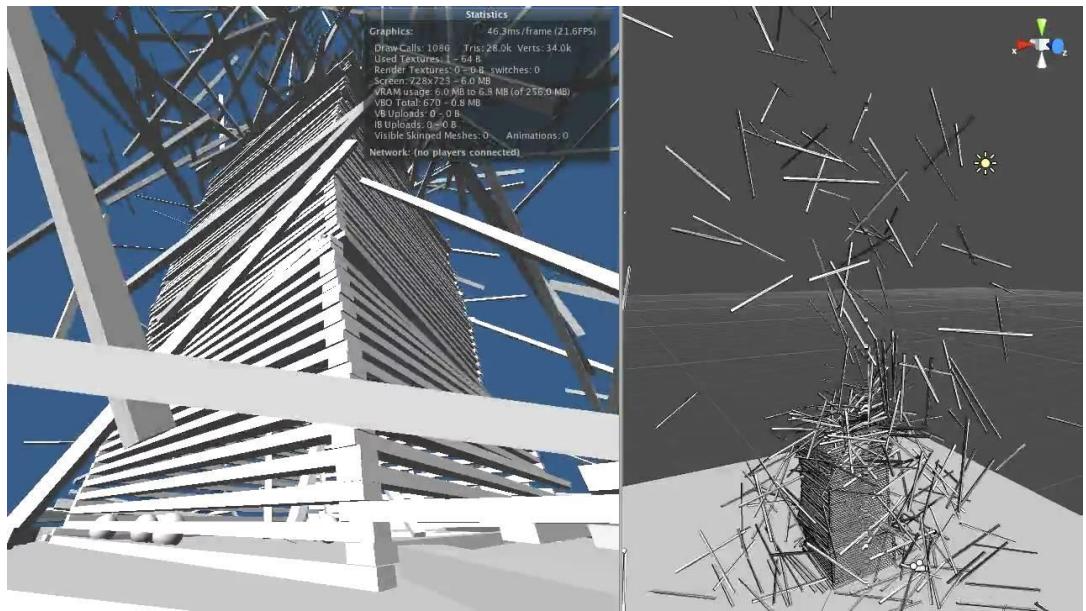
Figura 9 - Dispositivi di input durante gli eSport

L'Audio Manager fornisce e gestisce le risorse audio che verranno trasmesse all'utente tramite speakers. Solitamente, per risorse audio si intendono colonne sonore, rumori,

voci di personaggi e così via. Altra funzione dell'Audio Manager è l'applicazione di effetti su risorse audio che modifichino ed alterino il suono in ascolto [14].

Lo Scene Manager gestisce, sostanzialmente, i livelli del gioco e regola come determinate risorse debbano essere posizionate in uno spazio simulato 2D o 3D.

Il Physics Manager simula la fisica sulle componenti interattive del gioco, in modo da risparmiare al programmatore la scrittura di codice decisamente complesso. In sostanza, offre un comportamento fisico decisamente realistico (gravità, frizione, massa, ecc.) tramite poche linee di codice [13].



*Figura 10 - Simulazione della fisica in Unity 3D*

Lo Scripting Manager definisce la logica che offre un comportamento a ciò che compone il gioco [13]. Di solito, il Game Engine offre script precostruiti, ma nulla vieta che il Game Creator possa scrivere il proprio codice, utilizzando appropriati linguaggi di programmazione, al fine di personalizzare a proprio piacimento diversi aspetti del gioco.

Il Networking Manager permette la comunicazione tra due o più dispositivi, permettendo a molteplici giocatori di confrontarsi in un ambiente in tempo reale [13]. Riguardo tale componente in particolare, non è da dare per scontato che un Game Engine la includa di default: alcuni richiedono librerie esterne per implementare tali funzioni.



Figura 11 - Multiplayer tramite console Nintendo

Il Resource Manager è il sottosistema che tiene conto di tutti i materiali e le risorse che comporranno il videogioco, permettendone il caricamento a runtime nel caso siano necessari durante l'esecuzione. Per materiali e risorse intendiamo qualsiasi media che decori il videogioco, quindi modelli 3D, sprites 2D, colonne sonore e audio, animazioni e qualsiasi altro contenuto multimediale [14].

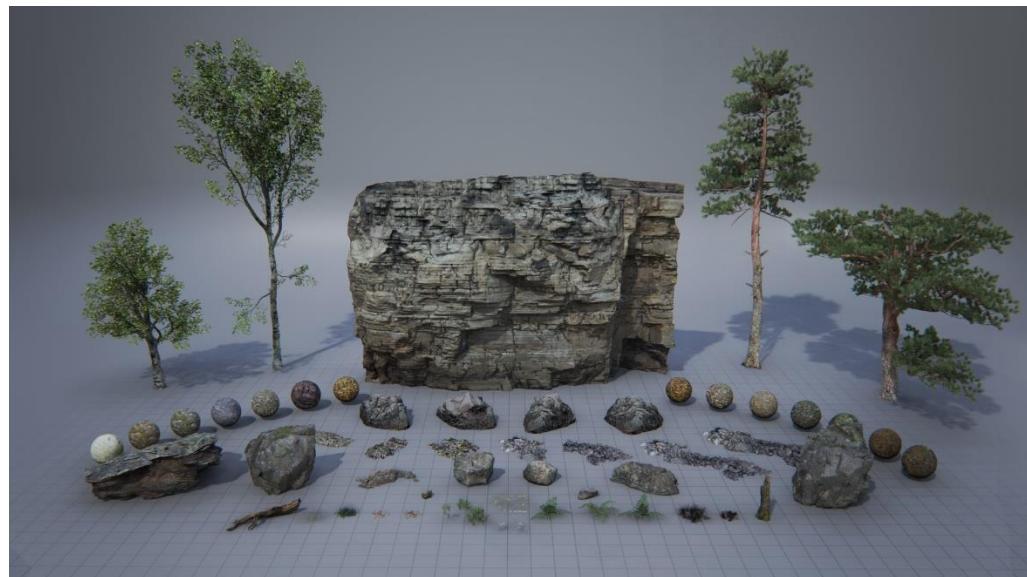


Figura 12 - Risorse ambientali di un gioco 3D

# Capitolo 3

## Tecnologie e Strumenti

### 3.1 Unity 3D

Unity 3D è un ottimo Game Engine per lo sviluppo videoludico, il quale offre molteplici funzionalità gratuitamente. Esso permette ai Game Creator di sperimentare le proprie idee, imparare e vendere giochi.

Unity permette la modellazione di videogiochi fondamentalmente tramite oggetti detti GameObjects, ai quali può essere assegnato un comportamento ben definito dal programmatore tramite script. Ogni GameObject è caratterizzato da una serie di attributi comuni utili alla sua disposizione nello spazio virtuale, come la posizione su coordinate e la sua rotazione; può essere definito e personalizzato tramite l'utilizzo di texture e materiali, script e componenti inerenti alla fisica [16].



*Figura 13 - Unity logo*

Unity, tramite la sua semplice interfaccia, permette al Game Creator di gestire tutte le sfaccettature del proprio videogioco. Ogni sottosezione dell’interfaccia si dedica ad una particolare caratteristica di ciò che si vuole modellare:

- Gerarchia: visualizza tutti gli oggetti di cui la scena corrente è composta in formato gerarchico; permette la gestione di GameObjects esistenti e la creazione di nuovi tramite apposito menu a tendina. I GameObjects sono raggruppabili anche in strutture padre-figlio, in modo da ereditare il comportamento di GameObjects generici;
- Finestra Progetto: offre un’anteprima, permette l’organizzazione e la gestione di tutti i contenuti della cartella Assets, quindi di tutte le risorse che compongono il videogioco. Inoltre, mostra tutte le librerie importate nel progetto;

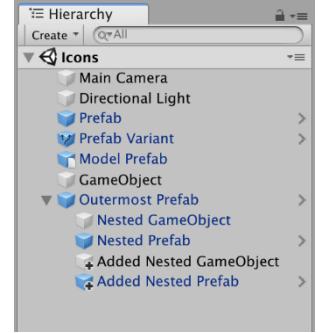


Figura 14 - Gerarchia

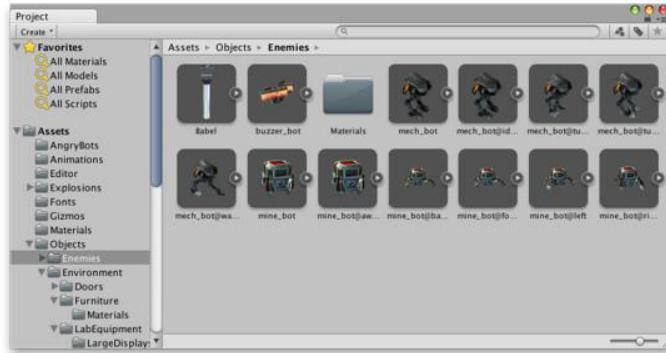


Figura 15 - Project Window

- Scene view: finestra dell’editor che consente la modellazione di un ambiente simulato, detto anche scena, interagendo dinamicamente con i GameObjects presenti, modellandone caratteristiche e dimensioni in maniera manuale tramite appositi strumenti;
- Game view: finestra che renderizza in tempo reale ciò che accade nella scena durante l’esecuzione del videogioco, dal punto di vista della camera di gioco attualmente attiva. In parole povere, simula nell’editor ciò che accadrebbe durante l’esecuzione della build del gioco;
- Inspector: permette la visualizzazione e la gestione di tutte le caratteristiche inerenti ad un preciso GameObject, quindi permette la gestione degli script, dei collider, della posizione, della rotazione e così via;

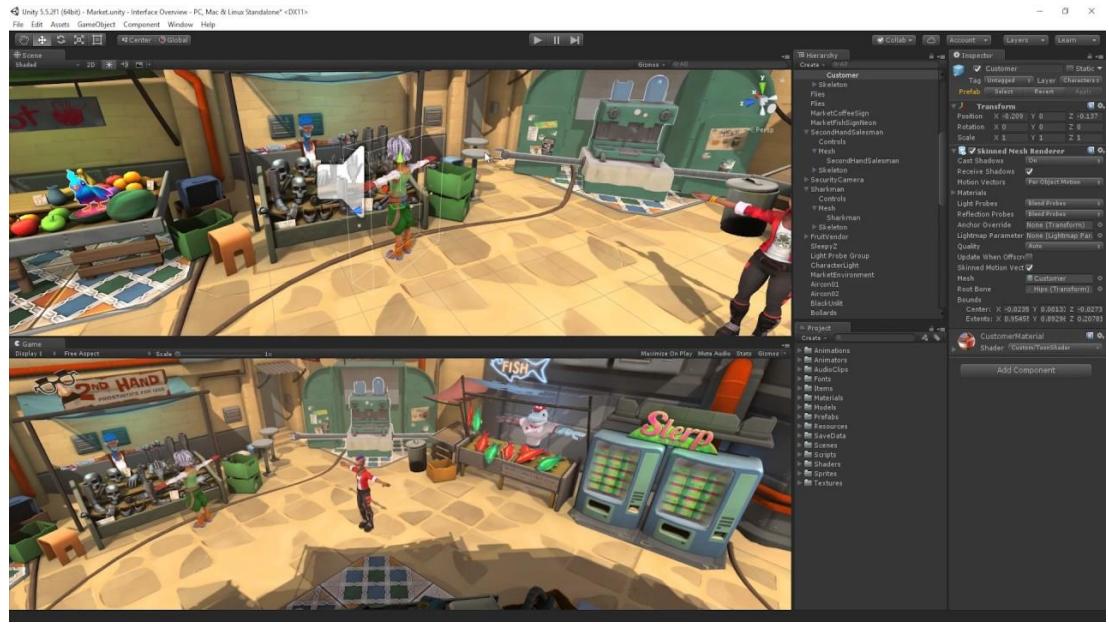


Figura 16 - Scene view, Game view e Inspector

- Asset Store: negozio integrato in Unity che permette l'acquisto o il download gratuito di risorse per il videogioco create dalla comunità [15].

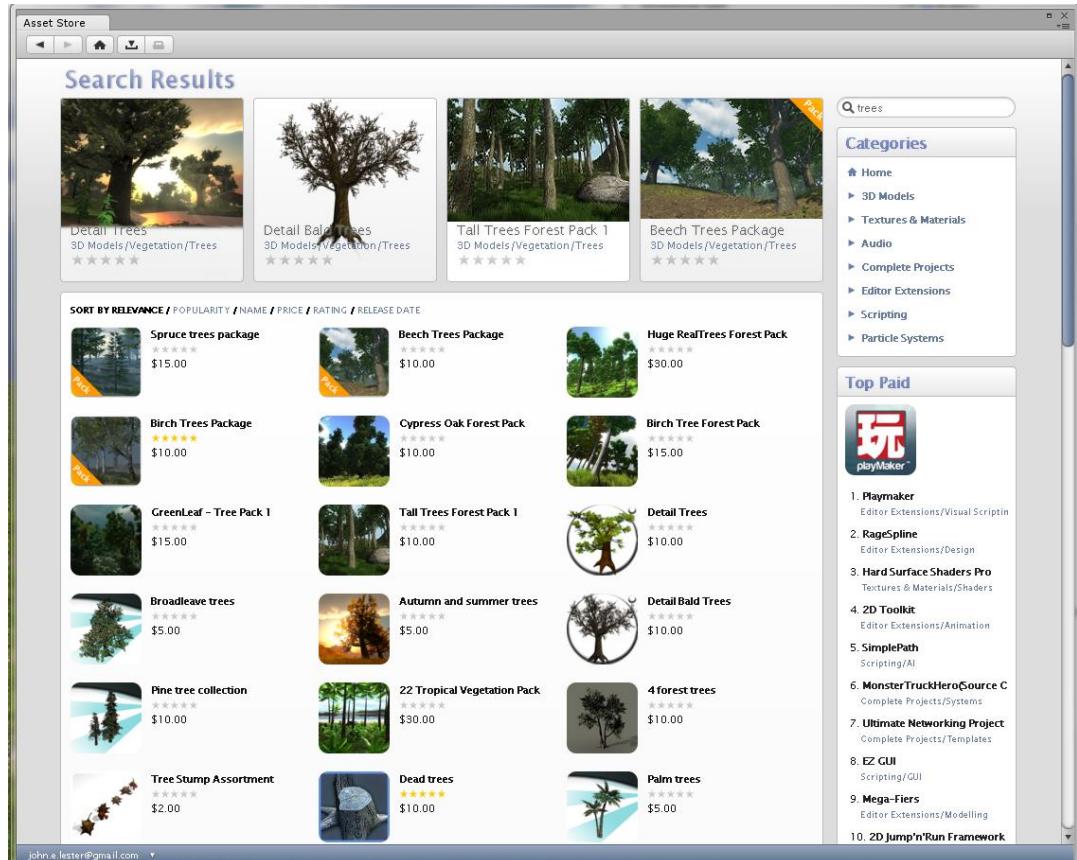


Figura 17 - Asset Store

### 3.1.1 Scripting e linguaggio C#

Lo scripting in Unity permette di definire un preciso comportamento per un GameObject nella scena. Lo scripting in Unity è differente dalla normale programmazione, difatti si tende a definire comportamenti e non l'infrastruttura di un applicativo da eseguire, in quanto è Unity che si occupa di compilazione e di esecuzione dei codici.

Gli script sono scritti in uno speciale linguaggio che Unity comprende, quale è C#, il quale è un linguaggio object-oriented, quindi orientato agli oggetti [17].

Ogni script è una classe che, in genere, eredita MonoBehaviour, in cui sono definite delle funzioni utili alla manipolazione del GameObject e delle sue componenti. Alcune delle funzioni presenti in MonoBehaviour sono `start()` e `update()`:

- La funzione `start()` esegue codice presente in essa alla creazione del GameObject;
- La funzione `update()` esegue costantemente il codice presente in essa.

MonoBehaviour non è l'unica classe ereditabile: è possibile estendere altre classi che forniscono comportamenti diversi basati su situazioni diverse. Un esempio è la classe Agent che offre funzionalità riguardanti il Machine Learning, oppure la classe NetworkBehaviour che offre funzionalità riguardanti il multiplayer.

## 3.2 ML-Agents

Lo Unity Machine Learning Agents Toolkit, conosciuto come ML-Agents, è un progetto open source che permette di simulare e istruire agenti intelligenti. Gli agenti possono essere istruiti tramite diversi tipi di apprendimento, in tal caso trattiamo il Deep Reinforcement Learning, con il supporto di una API scritta in Python.

Gli agenti istruiti possono essere utilizzati per diversi scopi: controllo degli NPC (personaggi non giocabili), testing automatizzati nelle build di gioco, in particolare simulazioni della realtà in ambienti simulati.

ML-Agents è utile sia ai Game Creator che ai ricercatori di Intelligenza Artificiale, siccome permette di addestrare agenti nel ricco ambiente di Unity, il quale è accessibile a chiunque, tramite una shell di comando [18].

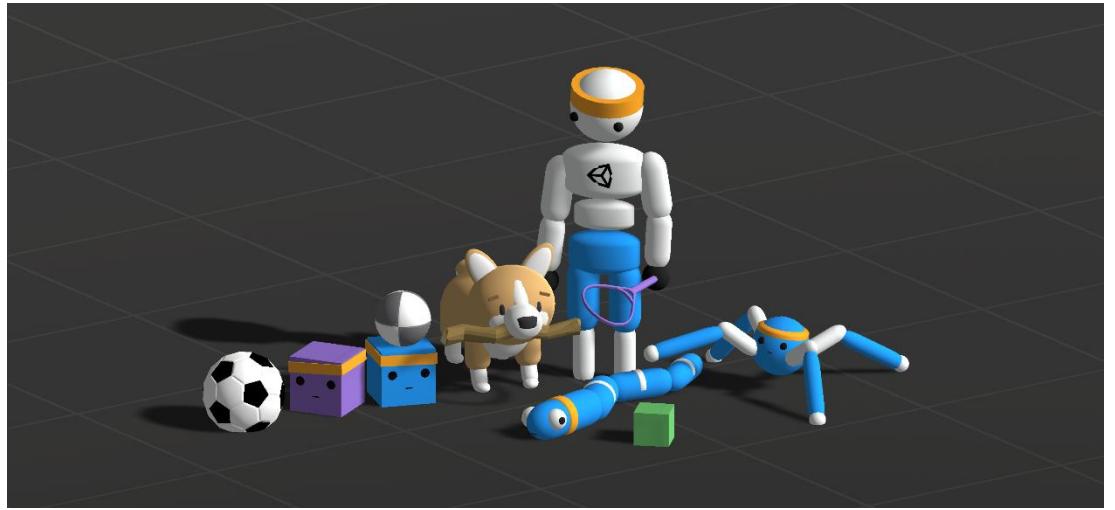


Figura 18 - ML-Agents Toolkit

### 3.2.1 Agente

L’agente è l’oggetto di scena che osserva e prende decisioni utilizzando il paradigma del Machine Learning tramite le informazioni ottenute interagendo con l’ambiente durante le sessioni di training.

L’agente fa uso di tre entità per modellare il suo comportamento e costruirne una Neural Network:

- Le osservazioni sono ciò che l’agente percepisce nell’ambiente circostante e possono essere valori numerici o visuali. L’ambiente esterno può essere osservato tramite l’ausilio di appositi sensori ed ogni tipologia di visualizzazione ammette specifici sensori, come specifiche camere integrate per le osservazioni visuali e sistemi basati su collisioni di raggi (RayPerception Sensor in Unity) con ostacoli per osservazioni di tipo numerico;
- Le azioni sono le decisioni che l’agente valuta giusto considerare in determinate situazioni. Possono essere continue o discrete: continue se vengono eseguite continuamente; discrete se vengono eseguite sporadicamente. La tipologia di azioni utilizzate dipende dalla complessità dell’ambiente circostante.
- Le ricompense, le quali sono scalari matematici utilizzati per far comprendere all’agente se ciò che sta eseguendo sia corretto o meno, in base al valore positivo o negativo, rispecchiando a pieno il paradigma del Reinforcement Learning.

Il comportamento dell'agente viene definito da script, il quale estende la classe Agent nella quale sono definiti una serie di metodi utili all'addestramento.

Inoltre, il GameObject agente è caratterizzato dalla componente Behaviour Parameters, la quale permette di definire la tipologia e il numero di azioni che l'agente dovrà eseguire in fase di training e di incapsulare una Neural Network prodotta dal training già concluso.

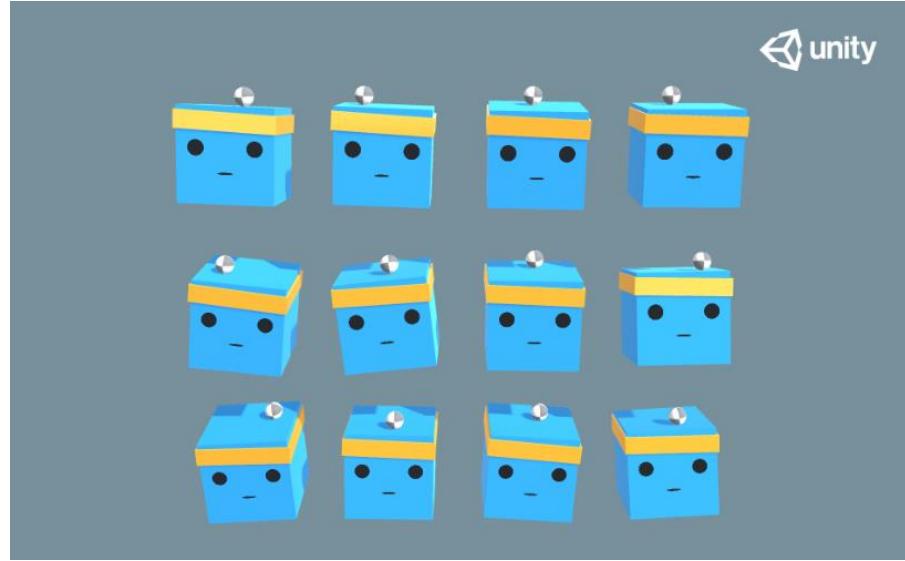


Figura 19 - Addestramento di molteplici agenti nel mantenere sopra la loro testa una pallina

Alcuni metodi della classe Agent sono i seguenti:

- **Initialize():**

contiene il codice utile all'inizializzazione dell'agente, quindi permette l'assegnamento di valori alla creazione di quest'ultimo;

- **CollectObservations(VectorSensor sensor):**

permette di definire i parametri da osservare, inerenti all'ambiente circostante, di cui si terrà conto durante l'addestramento, al fine di migliorare il comportamento dell'agente;

- **Heuristic(float[] actionsOut):**

permette di definire un profilo dell'agente manovrabile dal giocatore tramite comandi manuali. Questo metodo è molto utile durante la fase di debug e per altre tipologie di apprendimento, come l'Imitation Learning;

- **OnActionReceived(float[] vectorAction):**

tramite questa funzione, l'agente utilizza un vettore di azioni, composto da valori numerici, che influenzera le proprie scelte e decisioni. Che siano

giusti o sbagliati, dipende dai reward assegnati. Le azioni presenti in questo vettore possono assumere determinati valori in base alla tipologia di azione a cui si fa riferimento: se l'azione è continua, il valore numerico avrà valore  $k$  tale che  $-1 \leq k \leq 1$ ; se l'azione è discreta, il valore numerico è univoco per una determinata scelta che l'agente dovrà intraprendere;

- `OnEpisodeBegin()`:

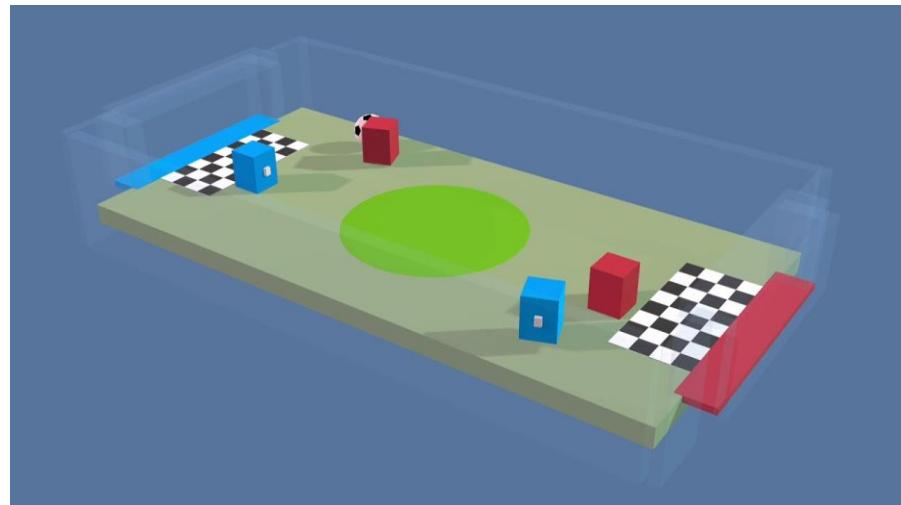
in questa funzione viene definito ciò che accade ad ogni inizio di un test, il quale viene anche chiamato episodio;

- `AddReward(float increment)`:

funzione utile per l'assegnamento di ricompense positive o negative, in base al valore increment passato come argomento. È preferibile che tale funzione venga richiamato al succedere di un certo evento, onde evitare che un determinato punteggio venga assegnato in ogni frame;

- `EndEpisode()`:

Determina la conclusione dell'episodio e si occupa del reset dell'agente, richiamando a sua volta il metodo `OnEpisodeBegin()`.



*Figura 20 - Agenti imparano a giocare a calcio in un piccolo campetto*

L'agente è caratterizzato da un Brain, il quale incapsula il processo che permette di prendere decisioni sulla base di osservazioni, azioni e algoritmi di apprendimento.

Esistono quattro tipi di Brain:

- Player: non è stata affrontata alcuna sessione di training e il giocatore può muovere liberamente l'agente in modo da testare tutte le sue funzionalità;

- External: per addestrare un agente, il suo Brain dovrà essere impostato su External siccome permette di ottenere ricompense tramite appositi algoritmi di apprendimento. Tale tipologia di Brain permetterà all’agente di ottenere informazioni dalle osservazioni, le quali permetteranno di prendere decisioni in maniera autonoma;
- Internal: una volta conclusa la sessione di training, il Brain potrà essere impostato su Internal per interpretare i risultati del training e per far effettuare decisioni all’agente tenendo conto di ciò che ha imparato;
- Heuristic: è una tipologia di Brain sperimentale che permette al programmatore di intervenire sulle scelte dell’agente [18].

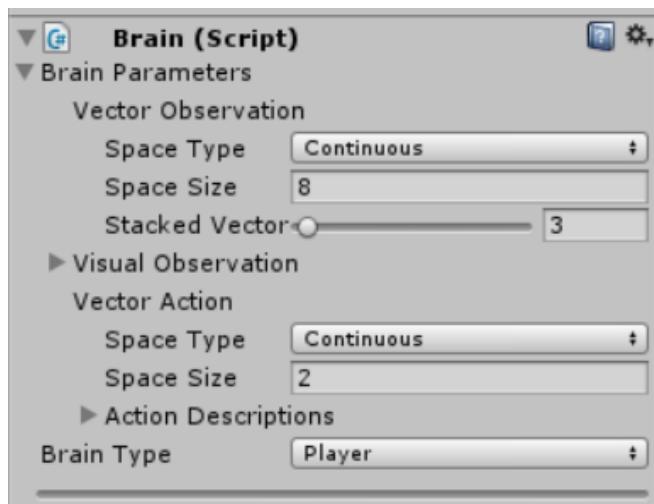


Figura 21 - Parametri del Brain

### 3.2.2 Linguaggio Python

Python è un linguaggio di programmazione ad oggetti ad alto livello di cui esistono diverse librerie dedicate alle applicazioni di Machine Learning. D’altronde, il toolkit ML-Agents si basa proprio su Python.

Il Machine Learning non è l’unico campo di cui Python si occupa, bensì è un linguaggio utilizzabile in moltissimi altri campi ed è in costante evoluzione.

Famoso anche per la sua particolare sintassi, la community di programmatore Python ha contrassegnato tale linguaggio da simpatici concetti:

- Beautiful is better than ugly;
- Simple is better than complex;
- Complex is better than complicated;

- Sparse is better than dense.

### 3.3 TensorFlow e TensorBoard

TensorFlow è una libreria open source utilizzata nel mondo del Machine Learning, inizialmente creata dall'organizzazione AI di Google. ML-Agents usa questa libreria per computare i risultati dei training e per analizzare i parametri dell'applicazione [19].

TensorBoard è un tool grafico che interpreta e mostra i dati inerenti a TensorFlow.

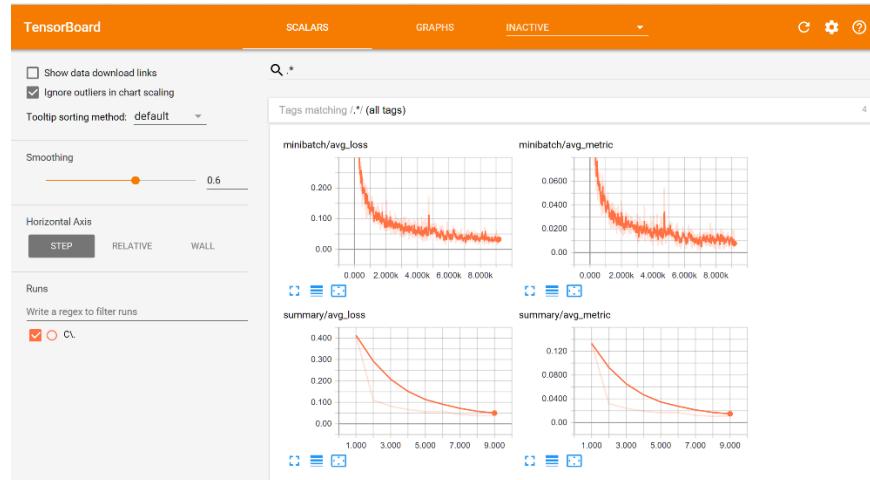


Figura 22 - Alcune sessioni di test su TensorBoard

TensorBoard è un potente strumento in grado di mostrare l'evoluzione del comportamento dell'agente tramite grafici.

## Capitolo 4

# Implementazione

AI Car Kineton è un progetto didattico, tenuto presso l’azienda Kineton, che simula tramite Machine Learning un’automobile dotata di sistemi ADAS per la guida automatica tramite ML-Agents, per il riconoscimento di ostacoli, per il riconoscimento del posto auto in cui parcheggiare e per il riconoscimento dei pedoni sulle strisce pedonali. Quest’ultimo caso viene trattato con dettaglio nel documento di tesi “Intelligenza Artificiale per il rilevamento dei pedoni in ambiente simulato” a cura di Carmine Ferrara.

In questo capitolo viene trattata la progettazione e l’implementazione del progetto, in modo da comprendere al meglio come un’automobile possa, tramite Reinforcement Learning, compiere manovre ed effettuare parcheggi senza fare incidenti.

## 4.1 Scenario in Unity

Per compiere sessioni di training e per vedere un'auto che cammina da sola, si sente la necessità di uno scenario 3D in cui simulare e di un'auto col ruolo di agente.

Lo scenario dovrà simulare una comune cittadina con una semplice strada, alla quale sarà affiancato il parcheggio con posti auto. L'idea, inizialmente, era quella di avere un'automobile che imparasse a riconoscere il percorso da seguire per poi imparare a parcheggiare; ergo, per lo scenario è stata progettata la seguente struttura:

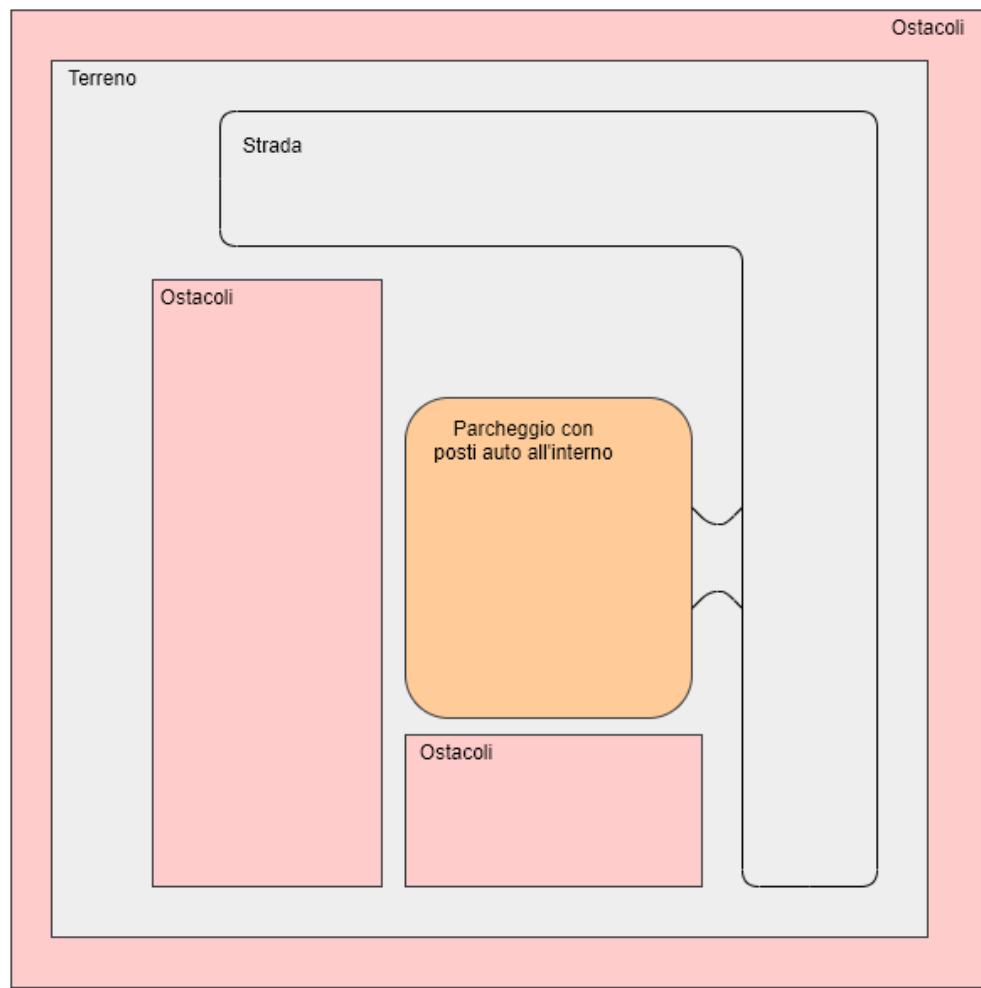
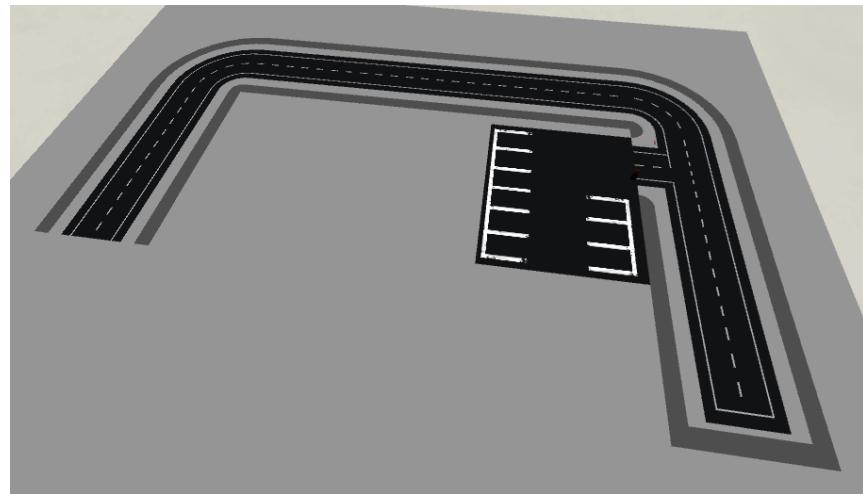


Figura 23 - Struttura iniziale della scena di gioco

Lo scenario sarebbe stato composto da un parcheggio con posti auto all'interno, abbastanza distanziati tra loro in modo da permettere adeguate manovre; da automobili statiche poste all'interno del parcheggio, in modo da complicare le manovre; da una strada inizialmente percorribile, dalla quale l'auto sarebbe partita per poi arrivare al parcheggio; una serie di ostacoli che, fondamentalmente, consistono nel decorare la scena.

Ovviamente ogni oggetto presente nella scena è provvisto di collision box, in modo tale che l'auto possa interagire con qualsiasi cosa e possa capire se l'interazione avvenuta sia giusta o sbagliata. Volendo porre un esempio, l'interazione con un'auto statica viene considerata un errore siccome raffigurerebbe un incidente, mentre l'interazione con un posto auto libero viene considerata corretta.

Lo sviluppo della scena è iniziato con la definizione dello scheletro di quest'ultima, quindi creando un terreno, la strada e l'area parcheggio.



*Figura 24 - Scheletro della scena*

Si continua poi con l'aggiunta di auto statiche all'interno del parcheggio e di una sua delimitazione, in modo tale che l'auto possa comprendere i confini entro cui risulterebbe all'interno del parcheggio: senza di essi, l'auto non comprenderebbe dove inizia e dove finisce l'area di parcheggio. Ad occhio è evidente, ma non bisogna dimenticare che l'auto vede tramite sensori e ragiona solo tramite essi, quindi deve poter riconoscere oggetti che delimitino il parcheggio.



*Figura 25 - Delimitazioni del parcheggio*

La scena è stata, quindi, decorata con recinzioni per il parcheggio, un piccolo giardino e qualche palazzo: il tutto circonda il parcheggio, quindi l'auto comprenderà le delimitazioni.

La modellazione della scena procede con l'inserimento di decorazioni, al fine di rendere il tutto più realistico. Ergo, il procedimento di creazione della scena si conclude con il suo raffinamento e la sua decorazione.



*Figura 26 - Scena di parcheggio completa*

Molte decorazioni inserite sono fuori mappa, al fine di coprire eventuali spazi vuoti che il giocatore potrebbe vedere.



*Figura 27 - Struttura della scena dall'alto*



Figura 28 - Automobili statiche poste all'interno del parcheggio

È stato detto che l'auto riconosce determinate situazioni e ambientazioni solo se i sensori riescono a percepire gli oggetti intorno all'automobile agente: come può, l'auto, riconoscere il posto auto in cui parcheggiare? Ebbene, il posto auto è caratterizzato da un oggetto invisibile percepibile solamente dal sensore di percezione ostacoli dell'agente, in modo da rendere distinguibile e osservabile il posto auto.

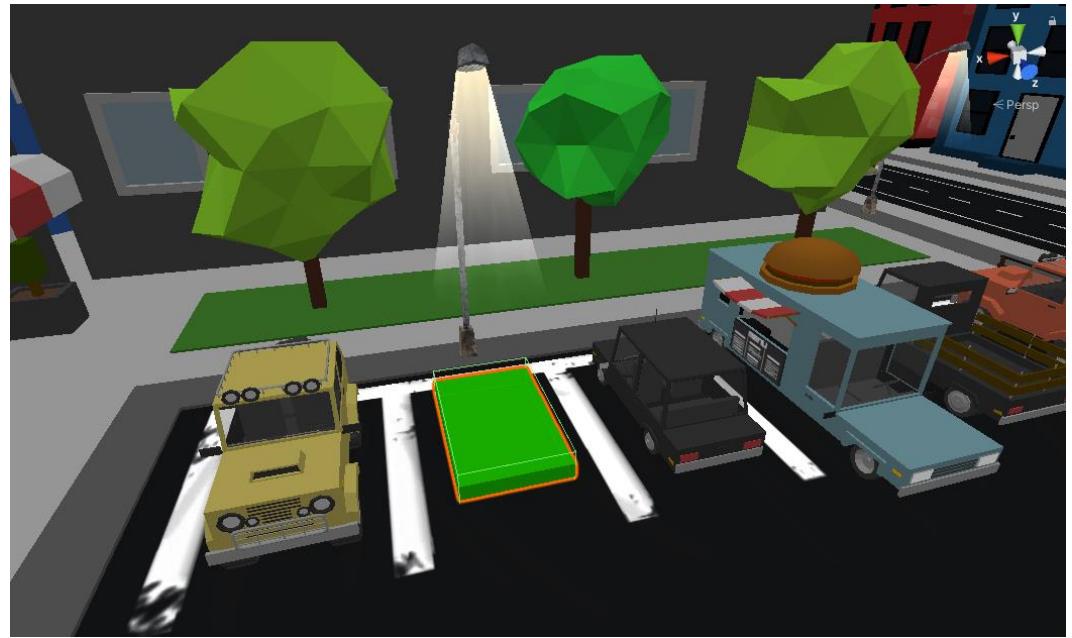


Figura 29 - Posto auto distinguibile dal sensore

Ora la scena è completa ed è pronta ad ospitare l'automobile con il ruolo di agente.

## 4.2 Modello di automobile virtuale

L'obiettivo consiste nell'addestrare un'automobile ad effettuare un parcheggio senza provocare incidenti: è necessario, quindi, avere un'auto dotata di sensori che la rendano intelligente.

Prima, però, si parte dal modello di una semplice e comunissima automobile.

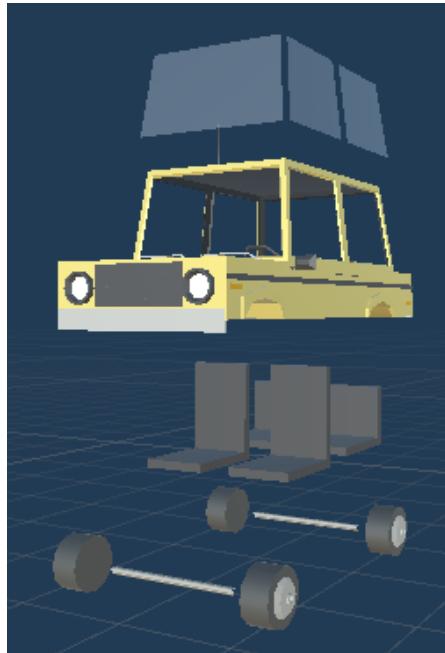


Figura 30 - Struttura auto

L'automobile presa in considerazione sarà composta dalle più basilari componentistiche di un'auto reale, quindi ruote, sediolini, una struttura esterna e dei vetri. Per quanto tale struttura possa discostarsi dalla realtà, ciò che interessa è il comportamento di guida automatica assunto durante la fase di apprendimento: questo perché l'auto, per guidare in maniera autonoma, fa uso di sensori e del sistema di sterzo controllato dal software, il quale è sostanzialmente composto dalle ruote e dalle assi che le collegano, di cui l'auto virtuale ne è provvista. Piccola aggiunta decorativa sono le luci posteriori per la frenata.

Le ruote dell'auto possiedono una componente capace di simulare la fisica dell'auto, quindi che permetta all'auto di svoltare e che faciliti l'accelerazione e la frenata, risparmiando al programmatore la scrittura di codice più complesso del dovuto. Inoltre, l'automobile virtuale sarà dotata di uno script che permetterà all'utente di comandarla tramite tastiera come se stesse utilizzando un comodo volante.



Figura 31 - Sterzata dell'auto virtuale

L'automobile riceve comandi tramite tastiera e, di conseguenza, effettua delle azioni che possono essere accelerazione (tasto W o freccia sopra), frenata (tasto Q), retromarcia (tasto S o freccia giù) o sterzata (tasti A-D o frecce sinistra-destra).

Infine, all'automobile saranno legate alcune telecamere, le quali permetteranno all'utente di visualizzare il parcheggio da diverse prospettive, tra cui l'interno auto. L'auto è fornita di un sottosistema di gestione delle videocamere automatico, attivabile e disattivabile, il quale imposterà la videocamera attiva in base alla direzione dell'auto, fornendo una visualizzazione dinamica in tempo reale di ciò che accade.

#### 4.2.1 Sottosistemi di semplificazione della guida

L'automobile virtuale è dotata di due sottosistemi, trasparenti all'utente, che semplificano sia la guida manuale che la guida automatica e influiscono sulla tenuta su strada.

Il primo sottosistema, denominato come Steering Control, riguarda la velocità tenuta in curva: se l'automobile si attesta ad entrare in curva ad una velocità molto alta e inizia a sterzare, l'auto frenerà automaticamente, a prescindere dall'input del conducente. Tale sottosistema è stato ideato per evitare che il conducente possa tenere una curva troppo larga a causa dell'alta velocità, quindi evitando incidenti con l'ambiente circostante.

Segue un'illustrazione che mette a confronto le due situazioni, cioè quando l'auto entra in velocità nella curva con guida totalmente manuale e quando l'auto utilizza il sottosistema appena illustrato.

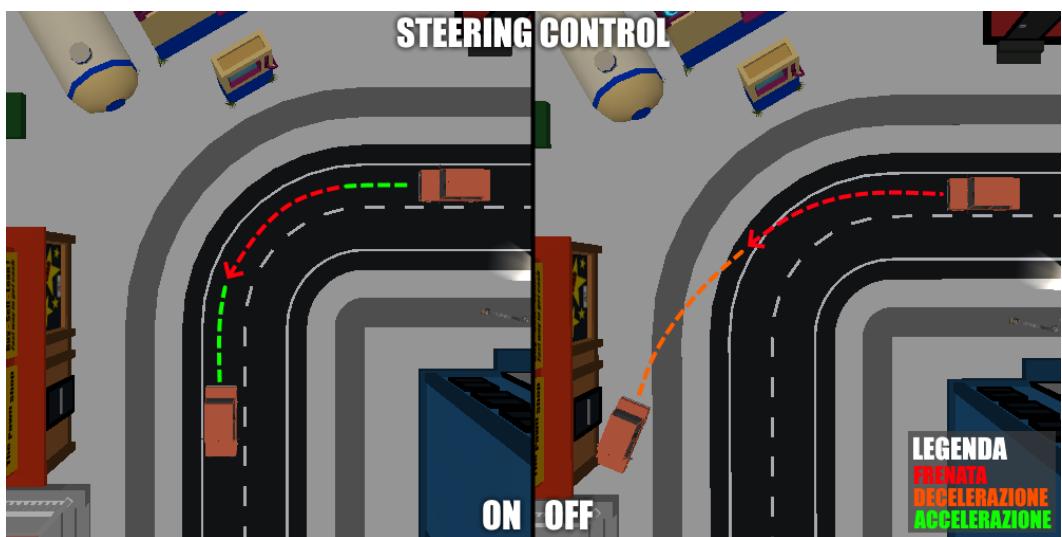


Figura 32 - Steering Control in curva

La velocità entro cui lo Steering Control agisce è impostabile manualmente in base alle situazioni: in un contesto urbano sarebbe utile impostare un limite inferiore abbastanza basso per la velocità; in un contesto extraurbano sarebbe, invece, utile impostare un limite inferiore più alto rispetto al caso precedente, siccome le curve saranno sicuramente più larghe e percorribili con velocità decisamente più alte.

Il sottosistema Steering Control utilizza un algoritmo molto semplice che permette la frenata automatica dell'automobile nel caso la velocità sia troppo alta e l'auto stia effettivamente sterzando. La forza applicata sul freno viene calcolata dinamicamente in base alla velocità dell'auto e viene denominata friction.

```
// Getting friction

float friction = getFriction();

// Speed check

if (actualSpeed > limitSpeed and

(wheelRotation == 35 or wheelRotation == -35)) {

    brake(friction);

}
```

Il secondo sottosistema, invece, riguarda la semplificazione dell'utilizzo della retromarcia. Si supponga una reale automobile con velocità  $v > 0 \text{ km/h}$ , per effettuare la retromarcia sarà necessario frenare e innescare la retromarcia: tale sottosistema consiste nell'utilizzo del pedale della retromarcia per rallentare (non per frenare, quindi creando meno attrito) fino all'innestamento automatico della retromarcia una volta che l'auto si ferma.



*Figura 33 - Tenuta su strada con il supporto dei sottosistemi*

#### 4.2.2 Automobile agente e sensori di guida

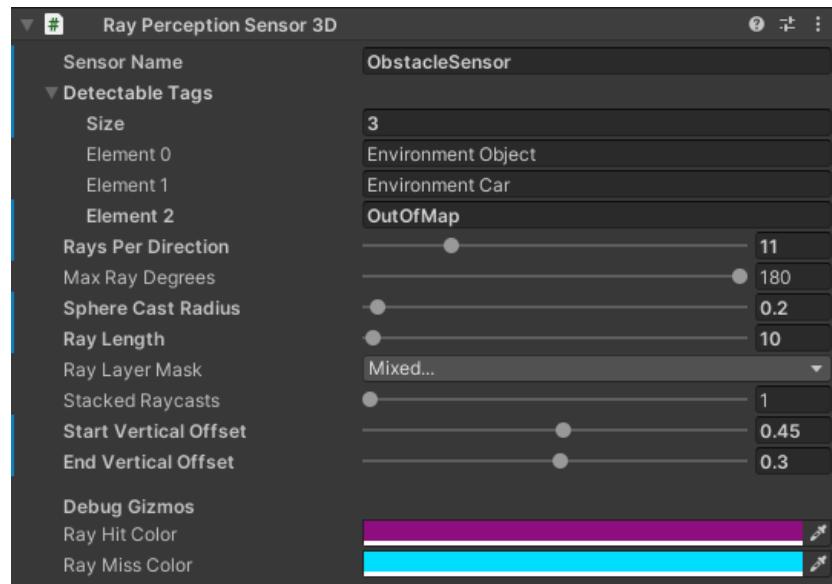
Avendo ora un’automobile manuale e completamente funzionale, è possibile farle assumere il ruolo di agente e prepararla alle sessioni di training montandole dei sensori. L’impostazione dei sensori è stato, probabilmente, il passaggio più complesso nella costruzione dell’automobile agente: sono state adottate e testate tre disposizioni di sensori, in modo da comprendere quale potesse essere la migliore.

I sensori utilizzati sono i RayPerception e consistono nell’emissione di molteplici raggi in grado di catturare le coordinate di collisione tra raggio e ostacolo, misurando le distanze tra agente e oggetti dell’ambiente. In sostanza, se un oggetto collide con un raggio, allora l’agente vedrà tale oggetto e ne registrerà la distanza. Se l’agente visualizza un oggetto dell’ambiente, il raggio cambierà colore.

I RayPerception hanno molteplici parametri personalizzabili, quali sono il numero di raggi da emettere, l’angolazione, la lunghezza, l’altezza, la colorazione, la grandezza delle sfere ed altro. Vediamo ora le più importanti:

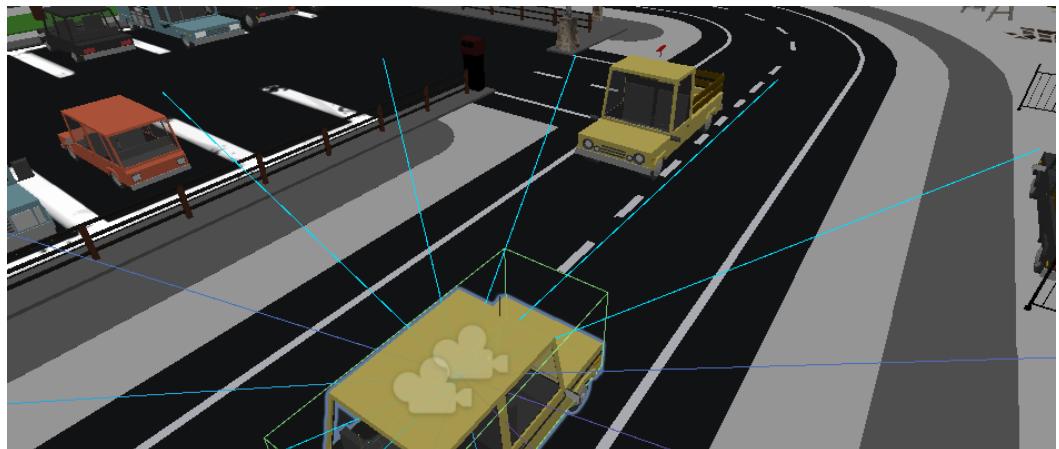
- Detectable Tags, stringhe che indicano i tag dei GameObjects da osservare in fase di addestramento. Ad esempio, l’agente dovrà tener conto delle auto statiche e del posto auto libero nel parcheggio: ergo, in Detectable Tags vanno inseriti i tag StaticCar e Destination;
- Rays Per Direction, indicano il numero di raggi ottimale al fine di garantire un’analisi ottimale di tutti i GameObjects specificati nei Detectable Tags;
- Max Ray Degrees, indica l’ampiezza del fascio di raggi (da 0 a 360 gradi), la quale incide molto sulla qualità del sensore simulato, siccome offre all’agente un campo visivo ampio secondo le necessità. Ad esempio, un’auto in un parcheggio dovrebbe avere tale parametro posto a 360 gradi siccome sarebbe preferibile avere una visuale completa dell’ambiente, mentre un’auto che deve far attenzione ai pedoni su delle strisce pedonali potrebbe avere un’angolazione minore, siccome non le servirebbe guardare nel lato posteriore;
- Ray Length, definisce la lunghezza dei raggi, la quale è fondamentale per la visualizzazione di determinati oggetti nell’ambiente simulato;
- Sphere Cast Radius, definisce la grandezza delle sfere utilizzate per ogni raggio;

- Ray Hit/Miss Color, definisce i colori dei raggi in caso di collisione o meno.

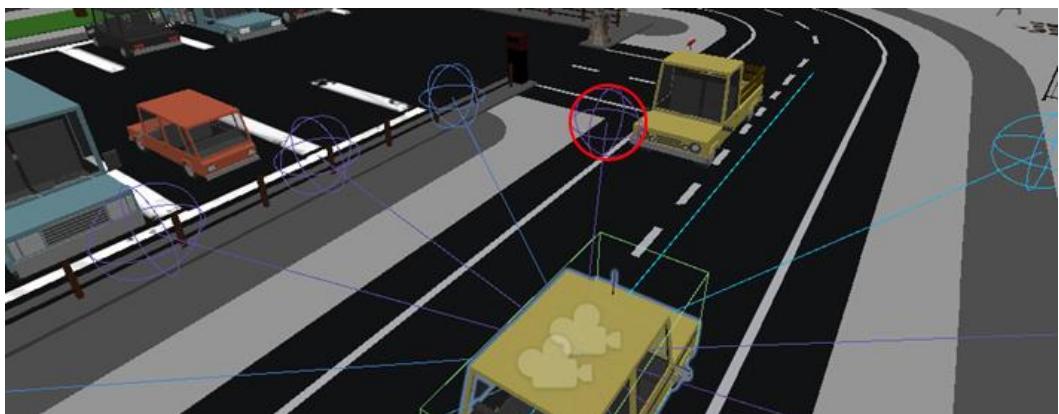


*Figura 34 - Impostazioni di un RayPerception Sensor*

Riguardo le sfere, ogni raggio può essere caratterizzato da una sfera, la quale copre lo spazio mancante tra un raggio e l'altro. Vedere un esempio rende sicuramente meglio.

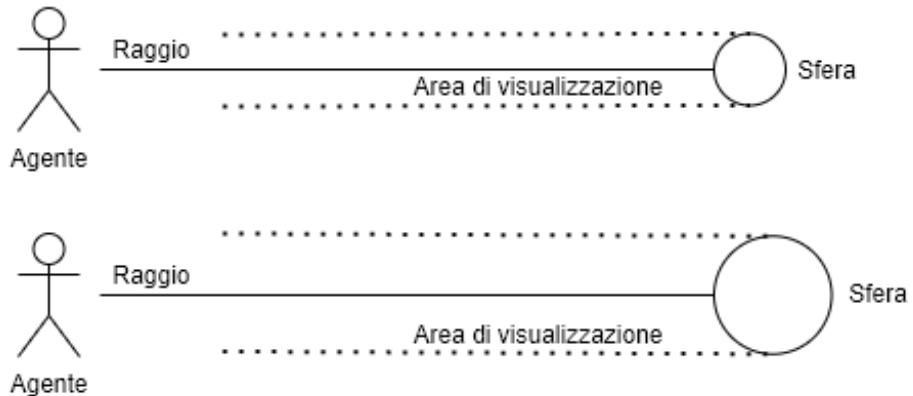


*Figura 35- Senza sfere, i raggi non vedono l'altra automobile*



*Figura 36 - Con le sfere, i raggi vedono l'altra automobile*

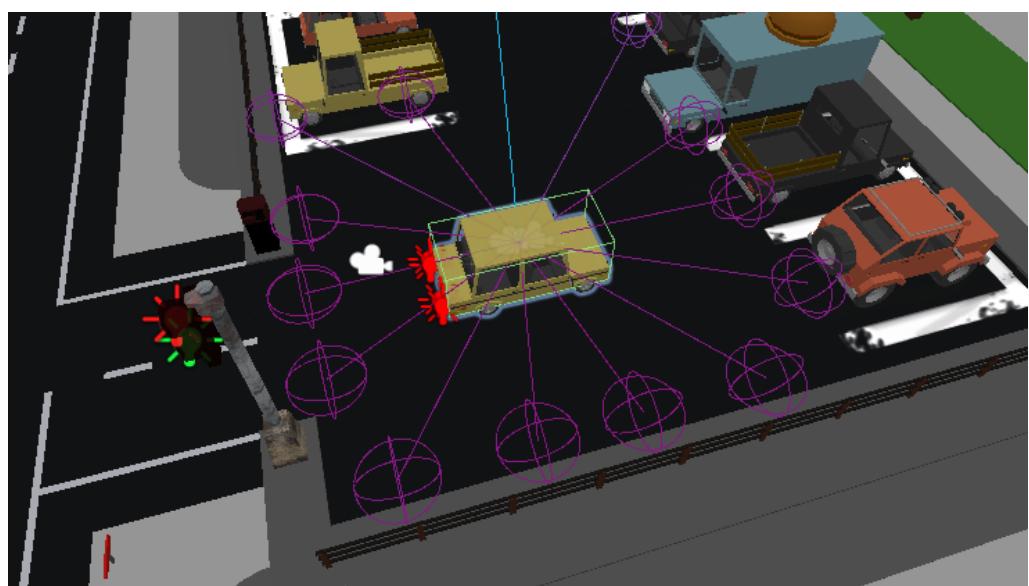
Le sfere evitano l'utilizzo di un numero elevato di raggi, in modo da individuare qualsiasi oggetto posto tra due raggi. Da ciò consegue che la sfera aumenta il diametro di visualizzazione del raggio.



*Figura 37 - La seconda sfera ha un diametro di visualizzazione più ampio*

Per ogni disposizione testata, sono state adottate tre configurazioni diverse: in ogni configurazione variano i valori dei parametri.

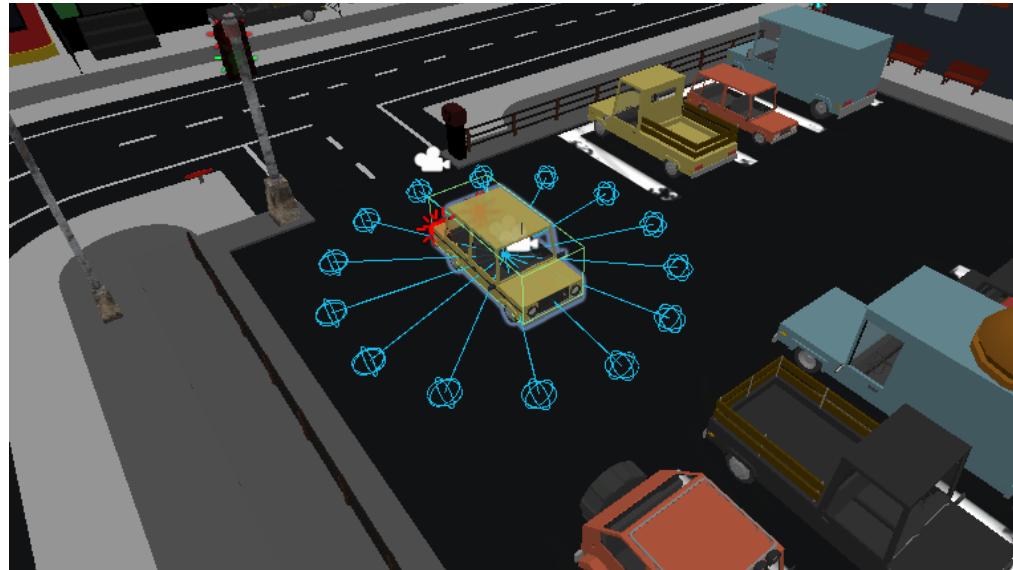
Nella prima configurazione sono stati adottati raggi relativamente lunghi con sfere di media grandezza: l'idea consisteva nel permettere all'auto di ottenere una visione completa del parcheggio, in modo tale che sin da subito l'agente comprendesse l'ambiente circostante.



*Figura 38 - Configurazione 1*

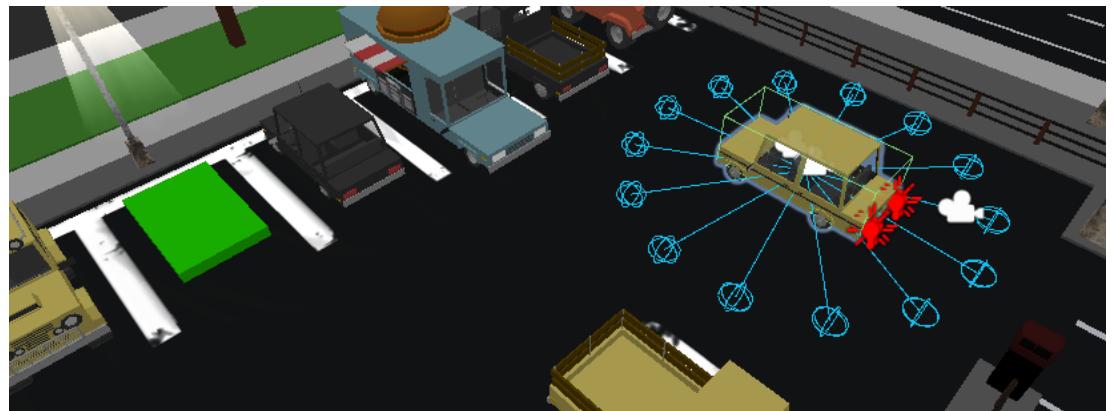
Il problema è che, come vedremo dettagliatamente nel prossimo capitolo, l'agente preferisce rimanere nella stessa posizione siccome nota troppi ostacoli e ha il timore di collidere con uno di essi.

Nella seconda configurazione sono stati adottati raggi relativamente corti con sfere molto piccole. Questa configurazione nasce come risoluzione del problema precedente, siccome in tal modo l'auto vede gli ostacoli in maniera ridotta.



*Figura 39 - Configurazione 2*

Ora l'auto può sentirsi decisamente più libera di camminare, senza la paura di collidere con ostacoli: difatti, il riscontro è stato più che positivo. Si presenta, però, un altro problema: in questo modo, l'auto non vede il posto auto siccome i raggi del sensore non lo individuano.



*Figura 40 - Posto auto troppo distante nella configurazione 2*

L'auto, quindi, cammina senza una destinazione e finisce, anche qui, con lo stare ferma per non collidere con gli ostacoli del parcheggio.

Da qui si è compreso il fatto che l'auto deve forzatamente vedere il posto auto e le altre auto per comprendere la destinazione. Come ben vedremo nel prossimo capitolo, tramite una semplificazione della scena si dimostrerà valida la prima configurazione,

ma presenterà un altro grave problema che darà vita alla terza configurazione, la quale fa uso di ben due sensori RayPerception con raggi notevolmente lunghi.

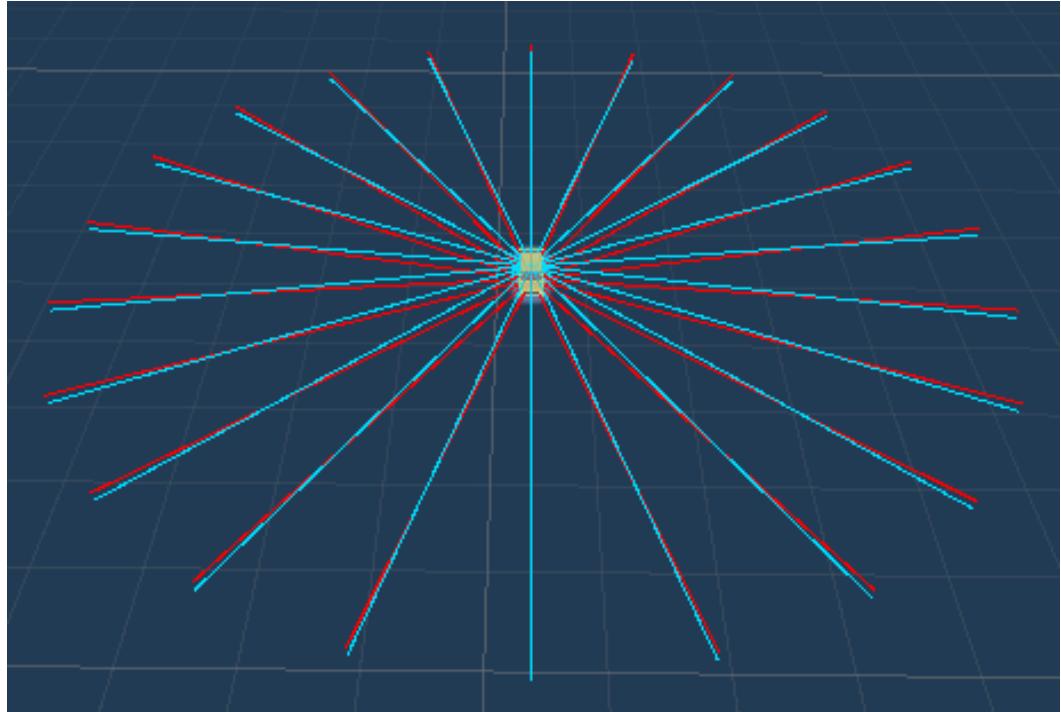


Figura 41 - Configurazione 3: doppio sensore

Il problema consiste nel fatto che, con un unico sensore, l'auto visualizza il posto auto ma non gli oggetti situati dietro quest'ultimo. Ciò comporta che in caso di parcheggio nel posto auto libero l'auto non vede determinati ostacoli (o li vede con visibilità ridotta) siccome i raggi sono bloccati sul posto auto posto davanti ad essa.

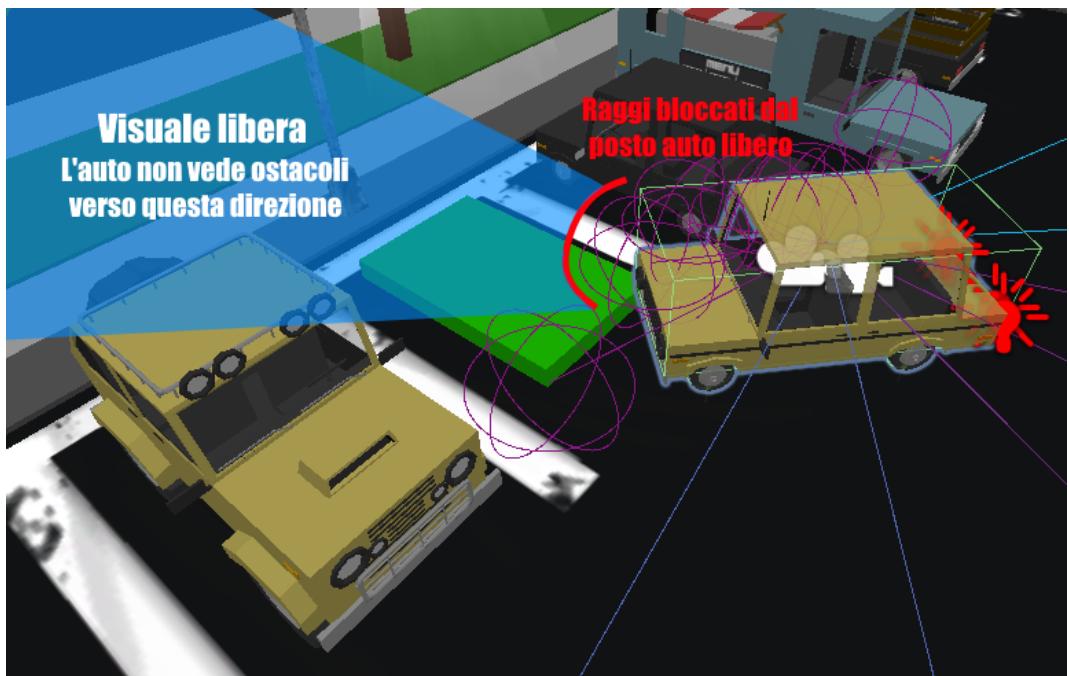


Figura 42 - L'auto non percepisce il palo vicino il posto auto libero

Si necessita, quindi, di differenziare gli oggetti riconosciuti tramite due appositi sensori RayPerception.

Un primo sensore si dedicherà al riconoscimento del posto auto libero, mentre un secondo sensore si dedicherà al riconoscimento degli ostacoli dell'ambiente. In tal modo, l'auto non si limita ad osservare solo il posto auto libero, ma anche l'auto statica posta al fianco ed il palo posto dietro.

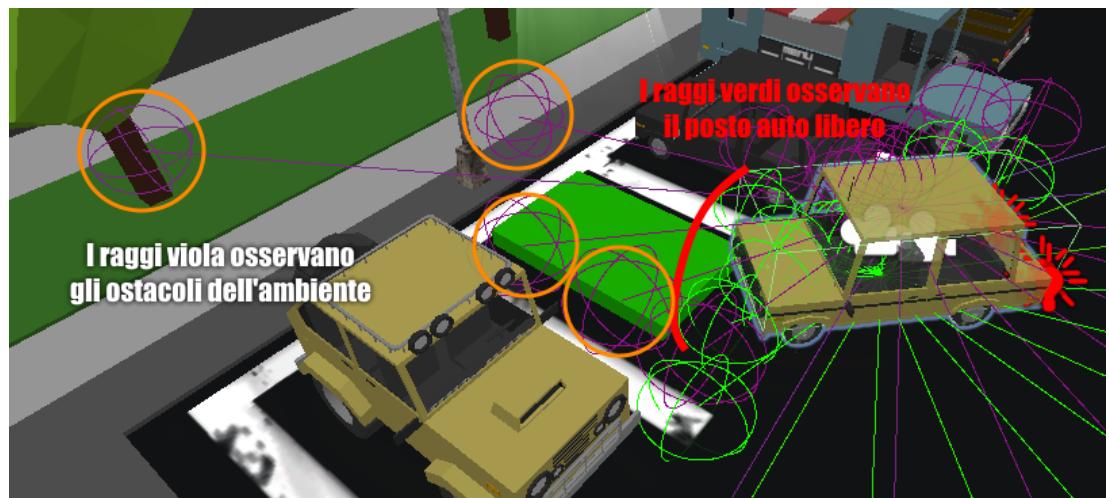


Figura 43 - L'auto percepisce gli ostacoli vicini al posto auto libero

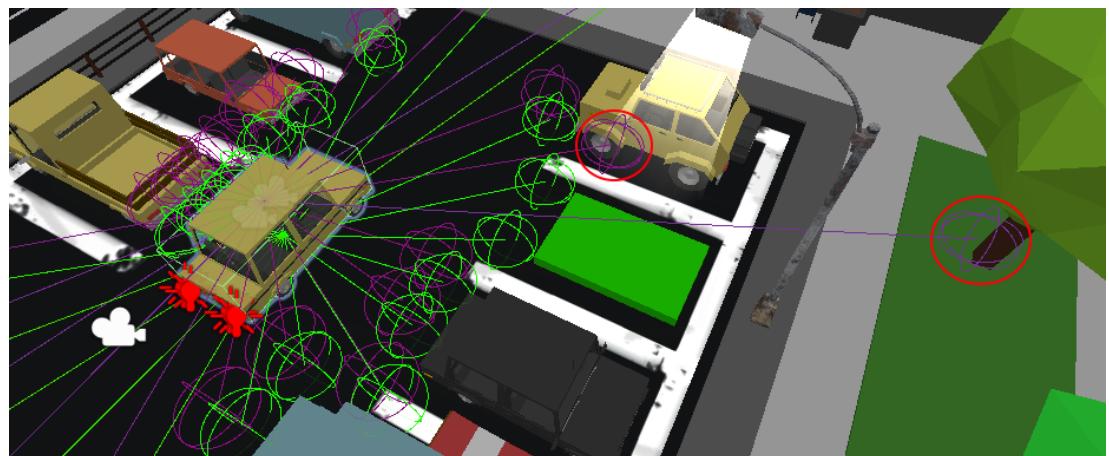


Figura 44 - Configurazione 3

In conclusione, la terza configurazione si dimostra essere la più affidabile siccome capace di individuare il posto auto e qualsiasi ostacolo vicino esso, in modo tale che l'auto riesca a prendere le giuste distanze da quest'ultimi.

Inoltre, la differenziazione tra sensore di rilevamento del posto auto e sensore di rilevamento ostacoli si avvicina ancora di più al modello di auto reale, siccome i sistemi ADAS differenziano i due sensori.

## 4.3 Educare l'agente

L'agente è ora equipaggiato da un buon sistema di guida, da un sensore di rilevamento ostacoli e da un sensore di rilevamento del posto auto libero: i sensori permettono all'agente di avere una buona percezione dell'ambiente simulato attorno a sé misurando le distanze da determinati oggetti tramite valori numerici. Tenendo conto della casistica del parcheggio, come fa l'agente a sapere che non deve collidere con le auto statiche, bensì deve collidere con il posto auto libero e rimanerci fermo all'interno? I sensori sono solamente uno strumento per avere informazioni circa la distanza da determinati oggetti nell'ambiente, non trasmettono informazioni circa cosa sia sbagliato fare o meno.

Spesso, quando si parla di Machine Learning, si paragona l'agente ad un bambino nato da pochi mesi: quest'ultimo osserva l'ambiente circostante tramite i suoi occhi, vede ogni oggetto posto attorno a sé, ma non sa come usare determinati oggetti e non sa se assumere un determinato comportamento sia giusto o meno. L'assunzione di un determinato comportamento sicuramente non dipende dalla vista del bambino, bensì dall'esperienza: ci vuole qualcuno che gli dica cosa sia giusto e cosa sia sbagliato nel tempo per insegnargli al meglio come comportarsi. Tale ragionamento si itera sull'agente, al quale va comunicato ad ogni azione intrapresa se sia giusta o sbagliata.

Per questo esiste un sistema di ricompense, basato sui movimenti dell'agente e sulle collisioni con gli oggetti dell'ambiente: ricordiamo che l'agente non sa di non dover collidere con auto statiche e altri oggetti, bensì deve comprenderlo in fase di apprendimento.

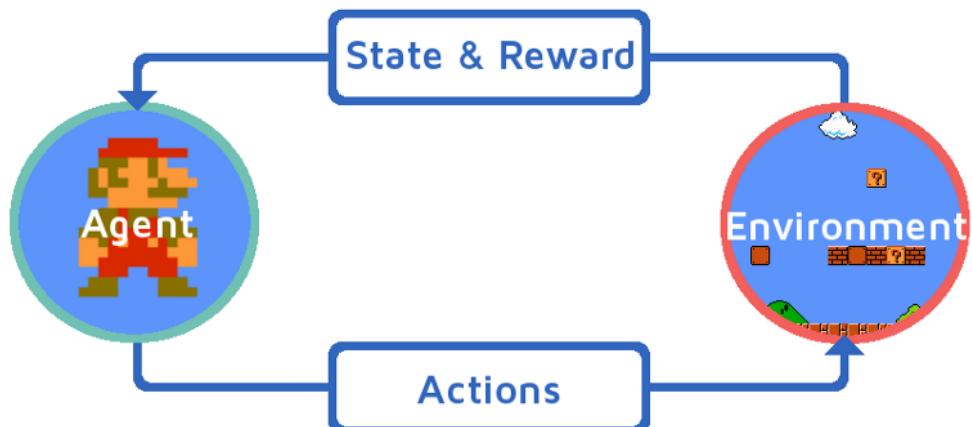


Figura 45 - L'agente interagisce con l'ambiente e comprende cosa sia giusto o sbagliato tramite reward

### 4.3.1 Sistema di ricompense

Come già anticipato, la tipologia di apprendimento adottata per l'agente è il Deep Reinforcement Learning, il quale si basa su un sistema di ricompense e su un sistema di riconoscimento tramite sensori.

Riguardo il sistema di ricompense, serve a far comprendere all'agente se una determinata decisione o scelta risulterà giusta o sbagliata, in base a diversi criteri dipendenti dall'ambiente circostante.

Essendo la ricompensa un valore numerico, si ragiona nel seguente modo:

$$\begin{aligned} \text{if } (\text{action is right}) &\rightarrow \text{reward} > 0; \\ \text{else} &\rightarrow \text{reward} < 0. \end{aligned}$$

Si passa, quindi, all'identificare qualsiasi azione commettibile dall'agente durante l'apprendimento è probabilmente l'operazione più complessa, siccome l'agente potrebbe continuamente presentare nuove scelte e decisioni ritenute banali e di cui non si è tenuto conto. In questa fase bisogna tener presente il fatto che l'agente proverà di tutto: non sapendo nulla dell'ambiente circostante, assumerà qualsiasi decisione e testerà qualsiasi casistica al fine di comprendere quanto più possibile circa l'ambiente circostante. Considerando il parcheggio, diamo per scontato che l'auto cerchi di parcheggiare, ma nulla le vieterà di uscire dal parcheggio e andare a farsi un giro per la strada, il che è una situazione che non deve accadere siccome l'obiettivo è parcheggiare. Bisogna tener conto di qualsiasi casistica possa presentarsi al fine di poter assegnare ricompense in base a qualsiasi azione l'agente compia.

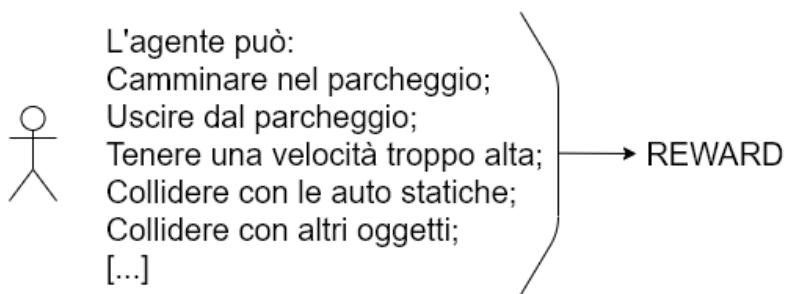


Figura 46 - Casistiche inerenti al comportamento dell'agente

È importante sapere che un valore medio di ricompense troppo basso rende l'agente meno intelligente, siccome pensa di non aver ben compreso come soddisfare un determinato compito e si spinge a provare nuove situazioni alternative al fine di riuscire a soddisfarlo. Invece, avere un valore medio di ricompense troppo alto lo rende

tropo sicuro di sé al punto da commettere errori senza pensare alle ricompense negative, le quali gli saranno indifferenti. Ciò è dovuto dal fatto che l'agente guadagna troppo tramite determinate azioni e perde troppo poco da altre azioni, il che fa sembrare la perdita futile. Il ragionamento che segue l'agente in caso di punteggio medio troppo alto è il seguente:

- Effettuo determinate scelte in una determinata situazione con un lower bound di punteggio medio pari a  $\lambda$ ;
- Siano  $\alpha$  e  $\beta$  i rispettivi guadagni netti per le scelte corrette e le scelte errate effettuate nell'attuale situazione, con  $\alpha > 0$ ,  $\beta < 0$  (ricordiamo che le scelte corrette hanno valore positivo, mentre le errate hanno valore negativo), sia  $bound > valore\ alto$ ,  $\alpha + \beta > bound$  e  $\alpha + \beta > \lambda$ , il nuovo punteggio medio  $\lambda'$  avrà sicuramente un valore maggiore del precedente  $\lambda' > \lambda$ , con

$$\lambda' = \frac{\lambda * n + (\alpha + \beta)}{n + 1}$$

dove  $n$  è il numero degli elementi che compone la media  $\lambda$ ;

- Siccome il punteggio medio dell'attuale situazione è molto più alto rispetto a quello della precedente situazione, significa che nel complesso ciò che ho fatto è da ritenersi giusto.

Un ragionamento del genere si verifica quando nella maggioranza delle situazioni si ha che  $\alpha + \beta > bound$  e  $\alpha + \beta > \lambda$  con  $\alpha > 0$ ,  $\beta < 0$  e  $bound$  indica un valore abbastanza alto, quindi quando il punteggio positivo è troppo alto e spesso maggiore della precedente media. Avere una crescita troppo alta del punteggio medio  $\lambda$  è controproducente, siccome l'auto comincerebbe ad abusare di determinati errori al fine di soddisfare l'obiettivo siccome il punteggio negativo viene considerato irrilevante.

Riprendiamo il parcheggio e poniamo un esempio. Poniamo che il punteggio medio dell'agente sia 120, che la collisione con un'auto assegna ricompensa -20, mentre la sistemazione corretta nel posto auto assegna 500; supponiamo che nel durante l'operazione di parcheggio l'auto collida con tre auto e riesca a sistemarsi nel posto auto:

$$\alpha = 500; \beta = -60$$

$$\lambda' = \frac{\lambda * n + (\alpha + \beta)}{n + 1} = \frac{\lambda * n + (500 - 60)}{n + 1} = \frac{\lambda * n + 440}{n + 1}$$

Siccome alla precedente media viene sommato un valore molto più alto, si avrà un grande incremento e l'agente continuerà a scontrarsi con le auto statiche siccome, nel complesso, la valutazione sarà più che positiva.

Altra problematica, quindi, sta nel decidere il valore della ricompensa da assegnare in base alla casistica presentata: anche questo passo è altamente importante, siccome bisogna bilanciare il valore da assegnare con la frequenza con la quale si presenta un determinato evento. Se un evento si presenta molto frequentemente, è bene assegnare una ricompensa minore rispetto ad un evento che si presenta frequentemente.

È buona prassi che il valore medio si tenga stabile, in modo da evitare situazioni simili a quella descritta.

Il valore delle ricompense è stato, quindi, assegnato tenendo conto dell'evento verificabile, con quale facilità e quanto potesse essere comune riscontrarlo.

Segue una tabella che mostra gli eventi di cui si è tenuto conto in primo momento, indicando il tipo di ricompensa allegata, la descrizione dell'evento, la frequenza (ogni quanto può capitare) e il premio (valore numerico assegnato al verificarsi dell'evento).

La legenda dei termini utilizzati nella tabella è posta dopo di essa.

Tipo	Descrizione	Frequenza	Premio
Negativo	Velocità eccessiva nel parcheggio	0-1/frame	-0.5
Negativo	Collisione con un oggetto di ambiente	0-1/EP	-100
Negativo	Collisione con un'automobile statica	0-1/EP	-100
Negativo	Agente uscito dal parcheggio	0-1/EP	-100
Positivo	Collisione con il parcheggio	0-1/SES	+250
Positivo	Ruota all'interno del posto auto	0 to 4/frame	+0.01
Positivo	Tutte le ruote all'interno del posto auto	0-4/frame	+0.08
Positivo	Bassa velocità all'interno del posto auto	0-1/frame	+spd/2
Positivo	Agente fermo nel parcheggio	0-1/EP	+500

*Se si parla di frequenza, con a-b si intende a oppure b volte; con a to b si intende qualsiasi valore compreso tra a e b inclusi; con EP si intende un episodio, quindi un test completo; con SES si intende una sessione di test; con il simbolo / si intende ogni (ad esempio, con 0-1/frame si intende 0 oppure 1 volta ogni frame). Infine, è presente uno speciale premio indicato con spd/2: si intende la velocità corrente dell'auto misurata in valore numerico diviso due.*

Come vedremo nel prossimo capitolo dedicato al training, gli eventi dichiarati nella tabella sono sufficienti per far assumere all'auto un comportamento adeguato, ma non per farle comprendere come guidare in maniera autonoma: l'agente non sa come muovere le ruote dell'auto in maniera autonoma, non capisce quando sia giusto sterzare verso destra o sinistra. Pertanto, si è sentita la necessità di sviluppare due sottosistemi di aiuto alla guida che impattassero solamente sull'assegnamento delle ricompense e non sul funzionamento dell'automobile.

#### 4.3.2 Sottosistema di ricompensa distanziale

Il sottosistema di ricompensa distanziale permette l'assegnamento di ricompense all'agente in fase di apprendimento in base alla distanza che intercorre tra l'agente e la sua destinazione, quindi il posto auto.

Tale sottosistema segue un algoritmo che permette di salvare la posizione raggiunta più vicina possibile alla destinazione e, nel caso l'agente si avvicini sempre di più, assegna una ricompensa positiva.

È un buon algoritmo di assegnamento di ricompense siccome incoraggia l'agente ad avvicinarsi ad un oggetto che magari in quel momento non conosce siccome non ci ha mai colliso.

Tale algoritmo è composto da una fase di inizializzazione e da una fase di registrazione della posizione e di assegnamento; quest'ultima, fa uso di una funzione che restituisce un vettore bidimensionale indicante la distanza di due oggetti.

```
// Initialization (one-time)
Vector2 bestDistance = getDistance(this, destination);

// Distance check (loop)
Vector2 actualDistance = getDistance(this, destination);
if (actualDistance < bestDistance) {
    bestDistance = actualDistance;
    AddReward(reward_value);
}
Debug.DrawLine(this, destination);
```

Vediamo l'assegnamento della ricompensa:

Tipo	Descrizione	Frequenza	Premio
Positivo	Avvicinamento al parcheggio	0-1/frame	0.02

Concludiamo con il sapere che tale algoritmo permette anche la visualizzazione di una retta nello spazio 3D che indica la posizione della destinazione.

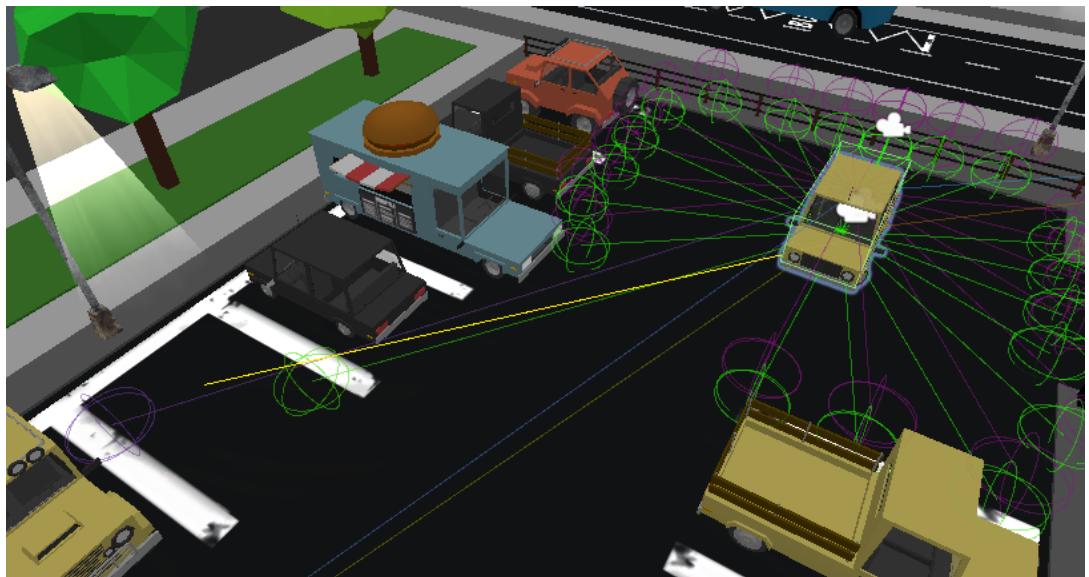


Figura 47 - La linea gialla indica la posizione della destinazione

#### 4.3.3 Sottosistema di ricompensa direzionale

Il sottosistema di ricompensa direzionale permette l'assegnamento di ricompense all'agente in fase di apprendimento in base all'angolo di sterzata relazionato all'angolo verso il quale è posta la destinazione.

Tale sottosistema non fa altro che dei controlli sull'angolazione delle ruote, confrontandole con l'angolazione di differenza tra l'agente e la destinazione. Se l'angolazione delle ruote si avvicina all'angolazione verso il quale è posta la destinazione, significa che l'agente è diretto verso essa e guadagna punti; in caso contrario, ottiene ricompense negative.

Prima di vedere l'algoritmo, è bene comprendere il ragionamento di background.

Poniamo di avere un'automobile agente le cui ruote possono sterzare al massimo di 35 gradi; sia  $\mu$  il grado di sterzata delle ruote in un determinato tempo  $t$ , sia  $\delta$  l'angolo di differenza tra l'agente e la destinazione con  $0 \leq \mu \leq 35$  e  $0 \leq \delta \leq 180$ .

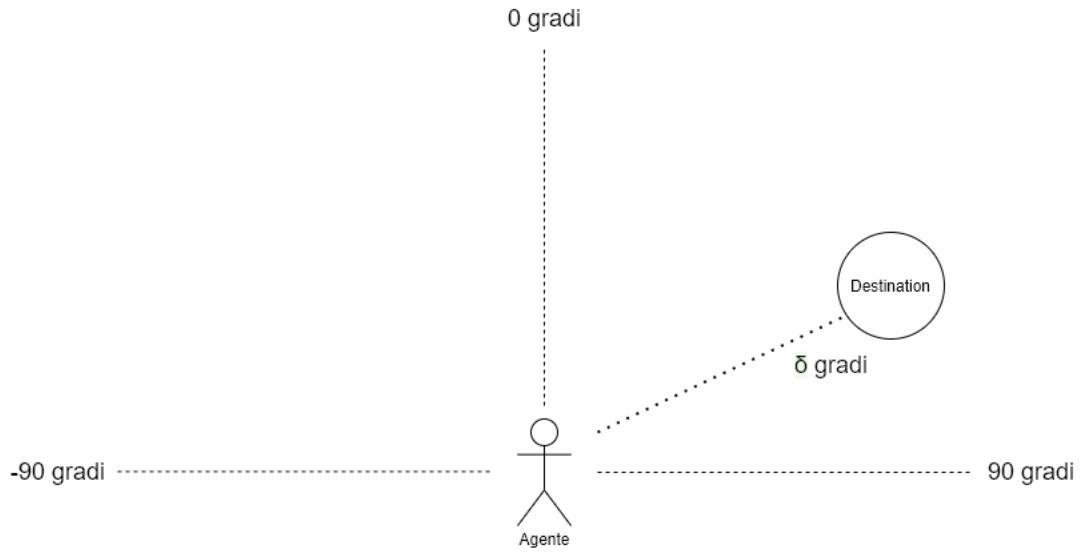


Figura 48 - Visualizzazione in gradi dell'agente

Il ragionamento di background consiste nel fare controlli su  $\mu$  in ogni frame, in modo da capire se l'angolo di sterzata delle ruote dirigerà l'auto verso la destinazione con angolazione  $\delta$ , in modo da farle comprendere la direzione giusta. Maggiori sono i gradi, maggiore dovrà essere l'angolazione delle ruote in direzione della destinazione; viceversa, minori sono i gradi, minore dovrà essere l'angolazione delle ruote.

Vediamo un esempio in cui poniamo che la destinazione si trovi a 70 gradi dall'automobile: l'automobile dovrà sterzare di 35 gradi per avvicinarsi sempre di più e ridurre la differenza, come mostrato nella seguente immagine.

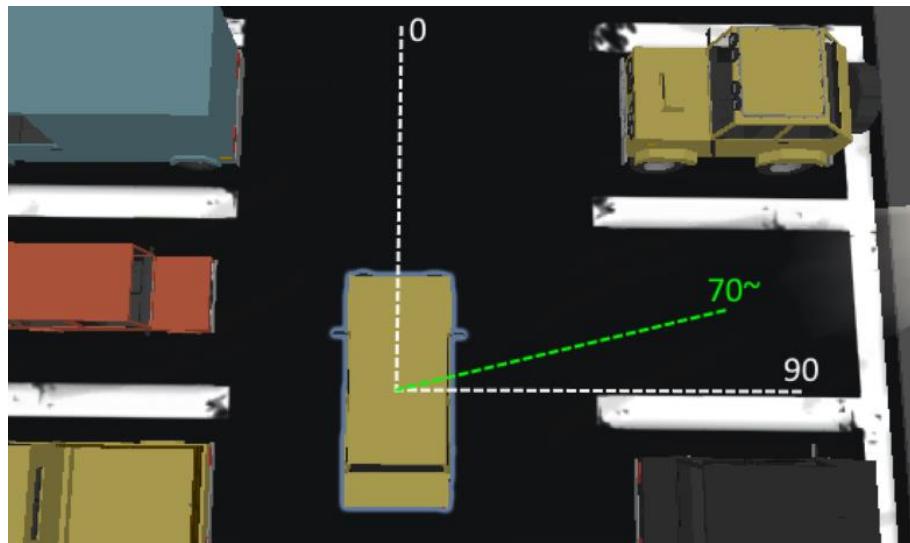


Figura 49 - L'agente si trova a 70 gradi dalla destinazione

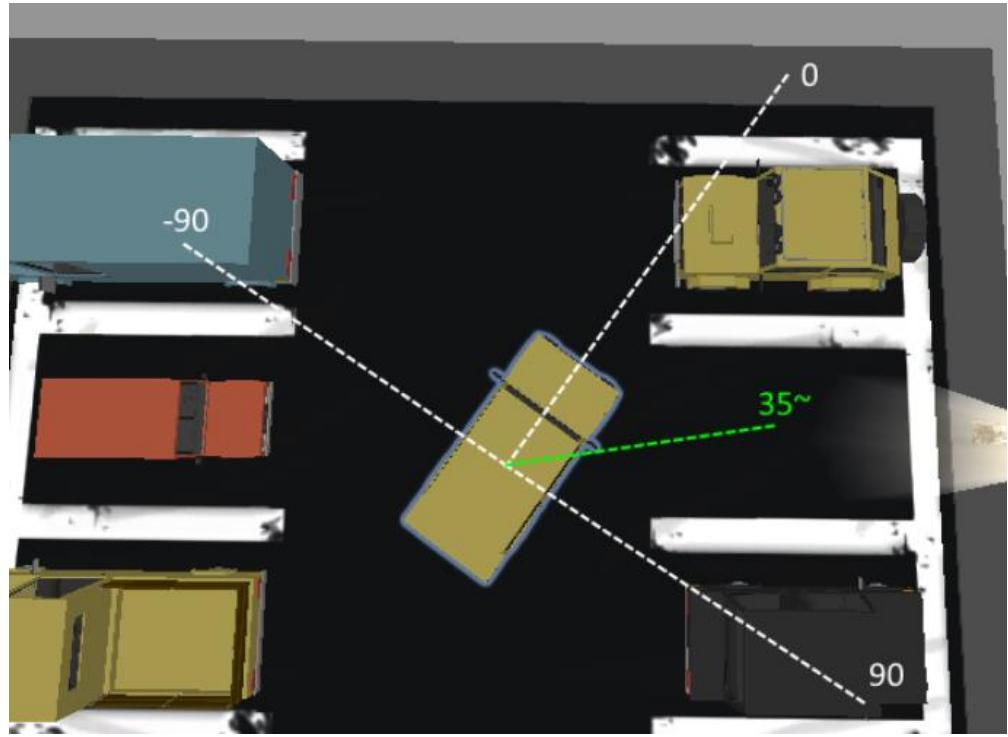


Figura 50 - L'agente svolta di 35 gradi e si trova a 35 gradi dalla destinazione

Viene, quindi, fatto prima un controllo su  $\delta$  e, in base al suo valore, viene controllato anche l'angolo di sterzata  $\mu$ . Nel caso appena visto, si è fatto un controllo del genere:

```
if ( $\delta > 35$  and  $\mu == 35$ ) -> AddReward(reward_value);  
else -> AddReward(negative_value);
```

Controllare, però, che  $\mu$  sia pari a 35 gradi non è corretto, siccome si potrebbero girare le ruote nel verso giusto ma non con 35 gradi: ergo, si controlla tramite un lower bound.

Segue il controllo corretto.

```
if ( $\delta > 35$  and  $\mu > 20$ ) -> AddReward(reward_value);  
else -> AddReward(negative_value);
```

Il ragionamento appena compreso va iterato anche nella sterzata opposta: con il controllo  $\mu > 20$  si sta controllando se le ruote siano sterzate di almeno 20.0001 gradi, quindi se ruotate verso destra; è necessario effettuare lo stesso controllo riguardo la sterzata verso sinistra, la quale farebbe assumere alle ruote un'angolazione  $\mu < 0$ .

Difatti, quindi, per ogni controllo sulle ruote è necessario effettuarlo su valori positivi di  $\mu$  e su valori negativi, in modo da includere entrambe le sterzate, se verso destra o sinistra. Il ragionamento appena compreso deve includere anche differenti angolazioni: abbiamo ragionato per un'angolazione  $\delta > 35$ , ma non basta. Iteriamo lo stesso ragionamento con altre angolazioni.

```

float assignment = 0;

// idle

if ((0 ≤ δ ≤ 3 and 0 ≤ μ ≤ 15) or
    (0 > δ ≥ -3 and 0 > μ ≥ -15))
    assignment = value;

// max difference angle = 35

if ((3 < δ ≤ 35 and 3 < μ ≤ 35) or
    (-3 > δ ≥ -35 and -3 > μ ≥ -35))
    assignment = value;

// difference angle > 35 or < -35

if ((δ > 35 and μ > 20) or
    (δ < -35 and μ < -20))
    assignment = value;

// assign reward

if (assignment > 0) AddReward(assignment)

```

Tale pseudocodice, però, è incompleto e assegna solamente ricompense positive: è necessario organizzare dei controlli in cui, in caso di angolazione delle ruote errata, viene assegnato un assignment negativo. Consideriamo il seguente controllo:

```

// difference angle > 35 or < -35 | Right case

if ((δ > 35 and μ > 20) or
    (δ < -35 and μ < -20))
    assignment = value;

// difference angle > 35 or < -35 | Wrong case

if ((δ > 35 and μ < 20) or
    (δ < -35 and μ > -20))
    assignment = value * (-1);

```

Il controllo dell'angolazione di differenza è identico, cambia piuttosto il controllo in cui le ruote hanno l'angolazione giusta: ora viene controllata l'angolazione errata. Nel caso, viene assegnata una ricompensa negativa.

Il codice, però, è ancora incompleto: i controlli appena visti non sono adatti alla retromarcia, la quale vuole si sterzi nell'angolo contrario rispetto all'angolo di differenza  $\delta$ . Vediamo questa situazione tramite raffigurazione.

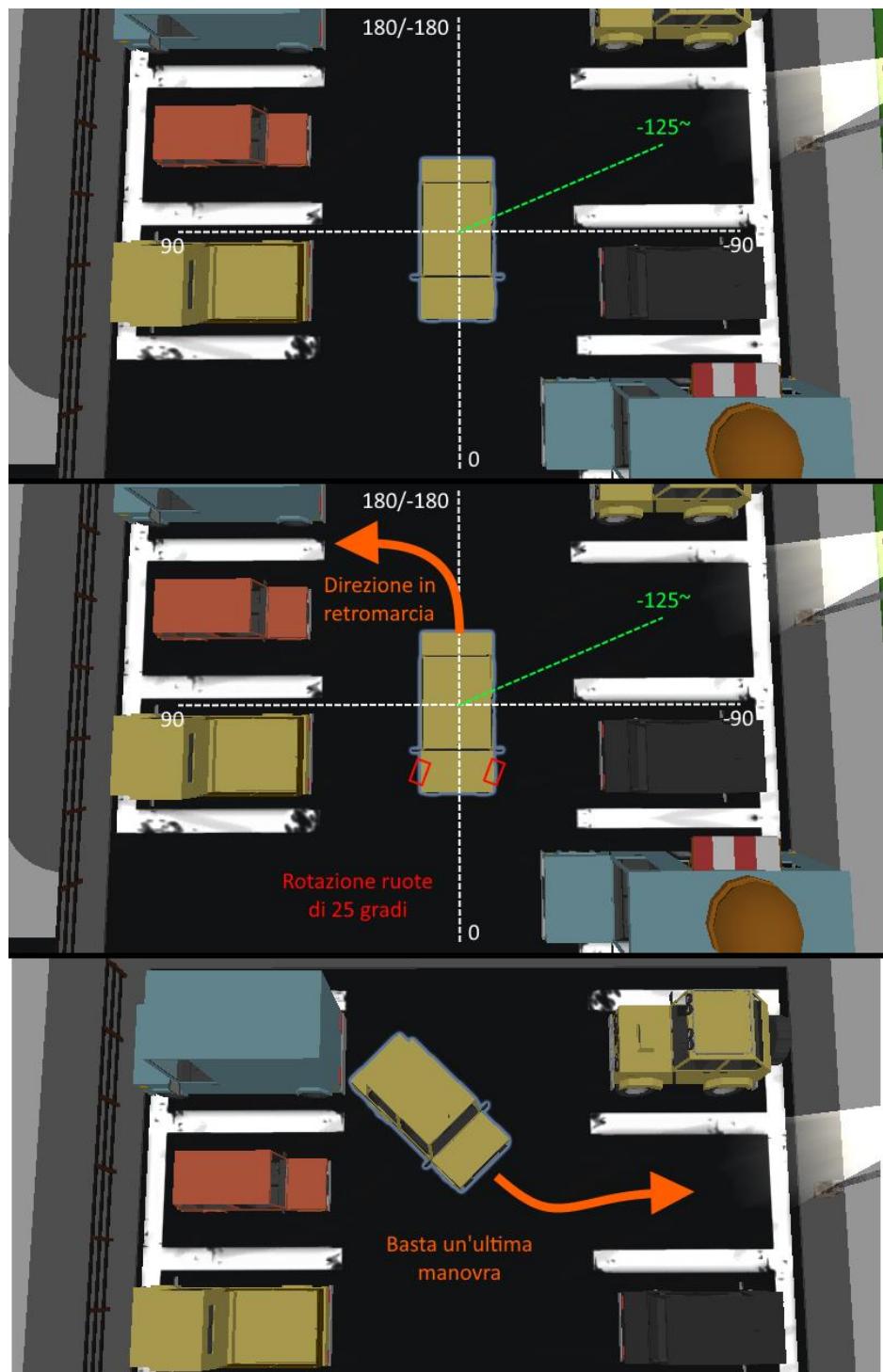


Figura 51 - Caso corretto di angolazione in retromarcia

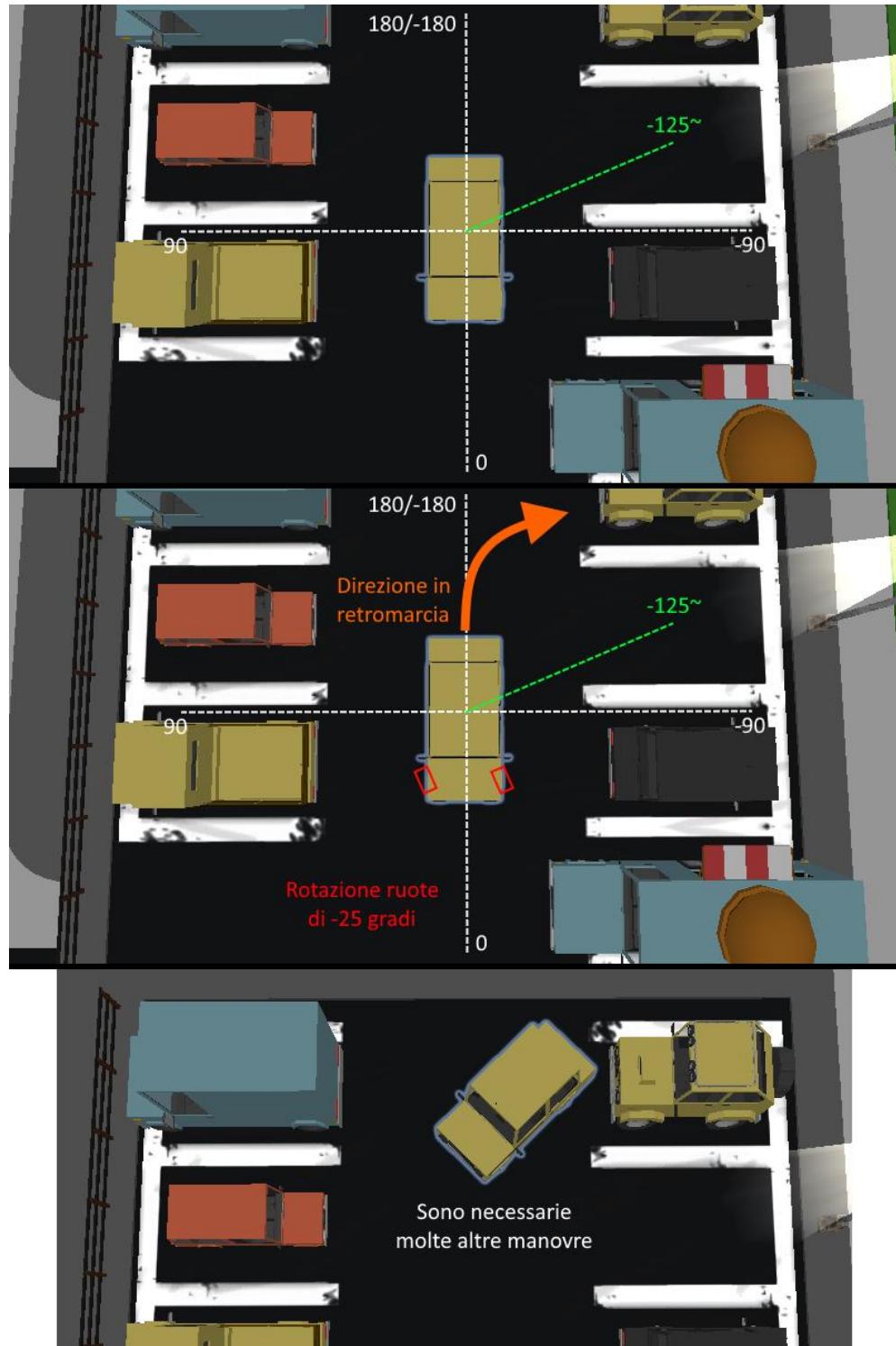


Figura 52 - Caso scorretto di angolazione in retromarcia

Se con il normale senso di marcia le ruote sterzavano verso l'angolo di destinazione, in retromarcia sterzeranno il contrario. Ciò significa che bisogna effettuare controlli anche riguardo il senso di marcia, se normale o retromarcia e, per ognuno di essi, effettuare i controlli in base alle angolazioni. In sostanza, per la retromarcia viene riutilizzato lo stesso codice di prima ma con gli angoli invertiti.

Segue il codice completo, ma è doveroso fare un paio di premesse:

- La funzione `getAngleMeasure(destination)` restituisce l'angolo di differenza  $\delta$  tra l'agente e la destinazione;
- Il parametro `steerAngle` indica l'angolo di rotazione  $\mu$  di una determinata ruota.

Iniziamo con l'analizzare il sistema di assegnamento sottoforma di funzione, la quale viene eseguita in ogni frame del gioco siccome tali controlli devono essere effettuati in maniera costante.

Richiede due parametri di cui uno facoltativo: il parametro `reward` indica il valore numerico indicato come ricompensa; il parametro `debug`, se posto a true, permette di stampare nel log del gioco se l'auto sta seguendo una direzione corretta o meno.

Tramite funzione `getAngleMeasure(direction)` viene ottenuto l'angolo di differenza  $\delta$  riutilizzabile nei controlli.

Viene posta una variabile `assignment` a 0: `assignment` assumerà un valore positivo se le ruote hanno l'angolazione corretta, mentre assumerà un valore negativo in caso contrario.

Fatti i dovuti controlli in base al senso di marcia, se normale o retromarcia, viene assegnata la ricompensa tramite `AddReward(assignment)`.

Infine, la funzione si conclude con il controllo del valore del parametro `debug`: se posto a true al richiamo della funzione, stamperà nella console di gioco se la direzione è corretta o meno. Inoltre, grazie alla funzione “collapse” della console di Unity, sarà possibile sapere quante volte sono stati stampati “Right direction” e “Wrong direction”, in modo da monitorare il grado di apprendimento dell'agente.

```
protected void directionAssignmentSystem(float reward, bool debug = false) {  
    //Getting angle  
    float carAngle = getAngleMeasure(connectedEndGame);  
    float assignment = 0;  
    //Checking by direction  
    if (actualDirection == Direction.Acceleration){}  
    else if (actualDirection == Direction.Backward){}  
    AddReward(assignment);  
    if (debug) {  
        if (assignment > 0) Debug.Log("Right direction");  
        if (assignment < 0) Debug.Log("Wrong direction");  
    }  
}
```

Figura 53 - Funzione del sistema di assegnamento intelligente in base alla direzione

Seguono rispettivamente i codici posti nei blocchi di codice IF inerenti alle direzioni.

```

if (actualDirection == Direction.Acceleration) {
    /* RIGHT CASES */
    //Straight check
    if ((carAngle < 3 && carAngle >= 0 && wc_fl.steerAngle <= 15f && wc_fl.steerAngle >= 0) ||
        (carAngle > -3 && carAngle < 0 && wc_fl.steerAngle >= -15f && wc_fl.steerAngle < 0))
        assignment = reward;
    //Car can steer of max 35* -> Right steer check
    if ((carAngle > 3 && carAngle <= 35 && wc_fl.steerAngle > 3 && wc_fl.steerAngle <= 35) ||
        //Left steer check
        (carAngle < -3 && carAngle >= -35 && wc_fl.steerAngle < -3 && wc_fl.steerAngle >= -35))
            assignment = reward;
    //Checking for angle > 35 or < -35 -> Right steer check
    if ((carAngle > 35 && wc_fl.steerAngle > 20) ||
        //Left steer check
        (carAngle < -35 && wc_fl.steerAngle < -20))
            assignment = reward;
    /* WRONG CASES */
    //Straight check
    if ((carAngle < 3 && carAngle >= 0 && wc_fl.steerAngle >= -15f && wc_fl.steerAngle <= 0) ||
        (carAngle > -3 && carAngle < 0 && wc_fl.steerAngle <= 15f && wc_fl.steerAngle > 0))
            assignment = reward * (-1);
    //Car can steer of max 35* -> Right steer check
    if ((carAngle > 3 && carAngle <= 35 && wc_fl.steerAngle < -3 && wc_fl.steerAngle >= -35) ||
        //Left steer check
        (carAngle < -3 && carAngle >= -35 && wc_fl.steerAngle > 3 && wc_fl.steerAngle <= 35))
            assignment = reward * (-1);
    //Checking for angle > 35 or < -35 -> Right steer check
    if ((carAngle > 35 && wc_fl.steerAngle < -20) ||
        //Left steer check
        (carAngle < -35 && wc_fl.steerAngle > 20))
            assignment = reward * (-1);
}

```

Figura 54 - Codice di controllo angolazioni per il senso di marcia normale

```

else if (actualDirection == Direction.Backward) {
    /* WRONG CASES */
    //Straight check
    if ((carAngle < 3 && carAngle >= 0 && wc_fl.steerAngle <= 15f && wc_fl.steerAngle >= 0) ||
        (carAngle > -3 && carAngle < 0 && wc_fl.steerAngle >= -15f && wc_fl.steerAngle < 0))
        assignment = reward * (-1);
    //Car can steer of max 35* -> Right steer check
    if ((carAngle > 3 && carAngle <= 35 && wc_fl.steerAngle > 3 && wc_fl.steerAngle <= 35) ||
        //Left steer check
        (carAngle < -3 && carAngle >= -35 && wc_fl.steerAngle < -3 && wc_fl.steerAngle >= -35))
            assignment = reward * (-1);
    //Checking for angle > 35 or < -35 -> Right steer check
    if ((carAngle > 35 && wc_fl.steerAngle > 20) ||
        //Left steer check
        (carAngle < -35 && wc_fl.steerAngle < -20))
            assignment = reward * (-1);
    /* OK CASES */
    //Straight check
    if ((carAngle < 3 && carAngle >= 0 && wc_fl.steerAngle >= -15f && wc_fl.steerAngle <= 0) ||
        (carAngle > -3 && carAngle < 0 && wc_fl.steerAngle <= 15f && wc_fl.steerAngle > 0))
            assignment = reward;
    //Car can steer of max 35* -> Right steer check
    if ((carAngle > 3 && carAngle <= 35 && wc_fl.steerAngle < -3 && wc_fl.steerAngle >= -35) ||
        //Left steer check
        (carAngle < -3 && carAngle >= -35 && wc_fl.steerAngle > 3 && wc_fl.steerAngle <= 35))
            assignment = reward;
    //Checking for angle > 35 or < -35 -> Right steer check
    if ((carAngle > 35 && wc_fl.steerAngle < -20) ||
        //Left steer check
        (carAngle < -35 && wc_fl.steerAngle > 20))
            assignment = reward;
}

```

Figura 55 - Codice di controllo angolazioni per la retromarcia

Concludiamo osservando il valore impostato per la ricompensa.

Tipo	Descrizione	Frequenza	Premio
Positivo	Ruote con angolatura corretta	0-1/frame	0.01
Negativo	Ruote con angolatura errata	0-1/frame	-0.01

## Capitolo 5

# Apprendimento

In questo capitolo vengono descritte tutte le sessioni di apprendimento affrontate al fine di istruire l’agente a parcheggiare e guidare in maniera autonoma. Per ogni sessione verrà fornita una panoramica circa il comportamento dell’agente, verranno illustrati i cambiamenti, le migliorie della scena e dell’agente, problematiche riscontrate e grafici riguardanti il livello di apprendimento dell’agente.

Ogni sessione è composta da molteplici test e ognuno di essi ha un proprio identificativo che segue la seguente struttura:

TypeName - SessionID - TestID

Il parametro TypeName avrà come valore di default “Parking”, mentre SessionID seguirà a sua volta due tipi di struttura: *FSk* e *LSk*, dove k è la numerazione della sessione. *FSk* e *LSk* sono due insiemi di sessioni che indicano due scelte progettuali adottate, tra loro differenti. Infine, il parametro TestID indica la numerazione del test appartenente ad una determinata sessione.

## 5.1 Introduzione alle sessioni FSk e LSk

Le sessioni di apprendimento svolte per far assumere all’agente un comportamento adeguato sono raggruppate in due insiemi mutuamente esclusivi tra loro: *FSk* ed *LSk*.

Gli identificativi *FSk* e *LSk* stanno per *FullSession\_k* e *LightSession\_k*, dove k riguarda la numerazione della sessione.

Ciò che distingue tali insiemi tra loro è la scelta di progettazione della scena e dell’agente adottata. Nelle sessioni *FSk*, l’agente è sottoposto all’operazione di training nella scena completa e con disposizione dei sensori secondo le configurazioni 1 e 2 viste nel capitolo 4.2.2 - *Automobile agente e sensori di guida*. Nelle sessioni *LSk*, invece, l’agente è sottoposto all’operazione di training in scene semplificate con difficoltà incrementante, con disposizione dei sensori secondo la terza configurazione, anch’essa illustrata nel capitolo sopra citato. Altra differenza è il numero di step di ogni test appartenente ad ogni sessione: uno step è un qualsiasi comando in input dato dal calcolatore all’agente al fine di comandarlo ed è la misura con la quale si misurano i test di ogni sessione. In sostanza, se un test è composto da K step, significa che durante quest’ultimo verranno effettuate K azioni da parte dell’agente. Ogni test delle sessioni *FSk* è composto da 500.000 step, mentre ogni test delle sessioni *LSk* è composto da 3.000.000 step.

C’è da fare anche un altro piccolo appunto, stavolta riguardante le ricompense: nelle sessioni *FSk* le ricompense avevano valori differenti rispetto a quelli mostrati nel capitolo 4.3.1 - *Sistema di ricompense*. I valori delle ricompense sono stati adattati nel corso dei molteplici test effettuati durante le sessioni. I valori mostrati precedentemente sono stati impostati per le sessioni *LSk*.

Ci sono altre piccole sfaccettature che differenziano i due gruppi di sessioni, ma le vedremo man mano che verranno illustrate le diverse sessioni.

## 5.2 Struttura delle sessioni

Ogni sessione sarà illustrata seguendo una determinata struttura, in modo da organizzare nella maniera più precisa possibile ogni differenza e cambiamento apportato all’introduzione di una nuova sessione.

La struttura con cui si illustrerà ogni sessione è composta dalle seguenti sezioni:

- Identificativo della sessione con numero di test;
- Panoramica della scena e di eventuali cambiamenti rispetto alla scena precedente, con annessa idea di background; panoramica riguardante il comportamento dell'agente durante e dopo l'addestramento, con eventuali problematiche riscontrate;
- Risultati (facoltativo), verranno mostrati solo per le sessioni con buoni riscontri.

## 5.3 Sessioni FSk

Le sessioni *FSk* trattano il training nella scena completa, quindi l'automobile agente ha a sua disposizione l'intera scena ed è libera di relazionarsi con essa come vuole. La scena non è, comunque, infinita ed è necessario apportare una piccola limitazione: nella strada che permette di uscire dalla città è posto un muro invisibile a cui l'agente non deve collidere, altrimenti uscirebbe fuori dalla mappa.



Figura 56 - Delimitatore OutOfMap

La collisione con tale muro invisibile assegna una ricompensa negativa.

Concludendo, l'automobile agente è caratterizzata da un unico sensore di rilevamento oggetti generico e non possiede il sottosistema di ricompense intelligenti mostrato nel capitolo 4.3.3 - *Sottosistema di ricompensa direzionale*.

### 5.3.1 Parking-FS1-X

Parking-FS1, sessione  $FSk$  composta da 3 test.

#### Panoramica e idea

Tale sessione di apprendimento avviene nella scena completa, quindi totalmente esplorabile, con punti di spawn dell'auto posti anche nella strada esterna.



Figura 57 - Punti di spawn

L'idea consisteva nel far imparare all'auto come guidare su una normale strada per poi entrare nel parcheggio e posizionarsi in un posto libero.

L'ingresso del parcheggio, inoltre, è caratterizzato di un muro invisibile (checkpoint) alla cui collisione dell'automobile agente le avrebbe assegnato una ricompensa positiva, per farle comprendere fosse la direzione giusta.

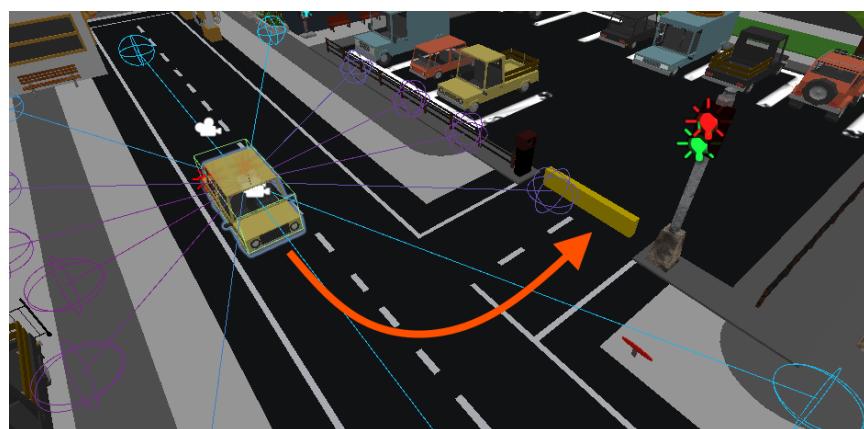


Figura 58 - L'agente vede il checkpoint

Nota: l'auto non è caratterizzata dai sistemi illustrati nei capitoli 4.3.2 - Sottosistema di ricompensa distanziale e 4.3.3 - Sottosistema di ricompensa direzionale.

## Comportamento dell'agente ed eventuali problematiche

Essendo il primo test in assoluto, sono stati dati per scontato molteplici aspetti: l'auto non sa come muoversi da sola, quindi non sa sterzare e raggiungere il checkpoint è estremamente complesso. Non colpendo il checkpoint, non sa che le assegnerà un punteggio positivo, quindi non è interessata a toccarlo. Può capitare che accada siccome l'auto, non sapendo guidare, muove le ruote a caso, ma sarebbe un caso isolato siccome non saprebbe come effettuare nuovamente la curva. La situazione diventa ancora più problematica se l'auto inizia il test vicino al pullman, siccome è necessario effettuare una curva senza saper guidare; inoltre, l'auto non vede nemmeno i confini della strada, quindi non sa che esiste una curva.



Figura 59 - L'auto non vede i confini della curva

L'auto non vede una destinazione e non vede delineazioni della strada, quindi cammina senza uno scopo uscendo anche dalla strada. Addirittura, nell'ultimo test della sessione, l'auto preferisce rimanere ferma onde evitare la collisione con oggetti di ambiente, i quali assegnano ricompense negative.

### 5.3.2 Parking-FS2-X

Parking-FS2, sessione *FSk* composta da 8 test.

#### Panoramica e idea

Date le difficoltà riscontrate nella precedente sessione in cui l'auto non riusciva a identificare la strada e la destinazione, si è deciso di ridurre temporaneamente l'area di gioco al solo parcheggio ed escludere momentaneamente la strada esterna.

L'idea è quella di far comprendere all'auto dove si trova il posto libero facendoglielo vedere tramite i sensori: i primi due test (1-2) pongono l'auto davanti al posto libero, in modo da allenare la guida automatica per una destinazione posta davanti ad essa; i successivi due test (3-4) pongono l'auto al lato sinistro del posto libero in modo da farle comprendere come sterzare; i successivi due test (5-6) pongono l'auto al lato destro del posto libero, in modo da farle comprendere come sterzare nel senso opposto. I rimanenti due test (7-8) pongono l'auto a 90 gradi del parcheggio.



Figura 60 - Punti di spawn dei test 1-6

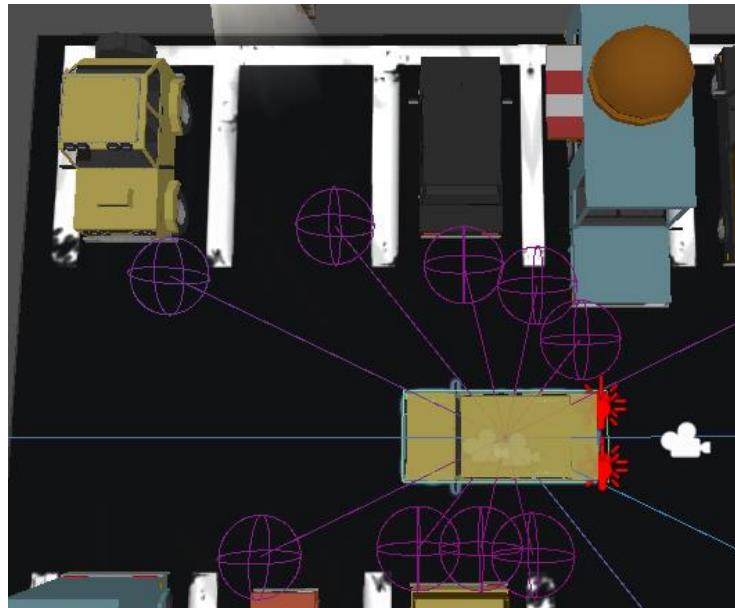


Figura 61 - Punto di spawn dei test 7-8

### Comportamento dell'agente ed eventuali problematiche

Tutti i test da 1 a 6 hanno portato buoni risultati: l'auto ha imparato a camminare in avanti e ad effettuare piccole sterzate, dirigendosi con successo verso il posto auto libero.

I problemi, piuttosto, si presentano con lo spawn dell'auto nei test 7-8: l'auto non riesce a sterzare con angolazione maggiore rispetto a quella con cui si è allenata nei precedenti test e preferisce rimanere ferma siccome vede troppi ostacoli tramite i sensori.

### 5.3.3 Parking-FS3-X

Parking-FS3, sessione *FSk* composta da un solo test.

#### Panoramica e idea

La terza sessione della serie *FSk* tende a risolvere il problema della precedente utilizzando la seconda configurazione del sensore, quindi raggi molto più piccoli.

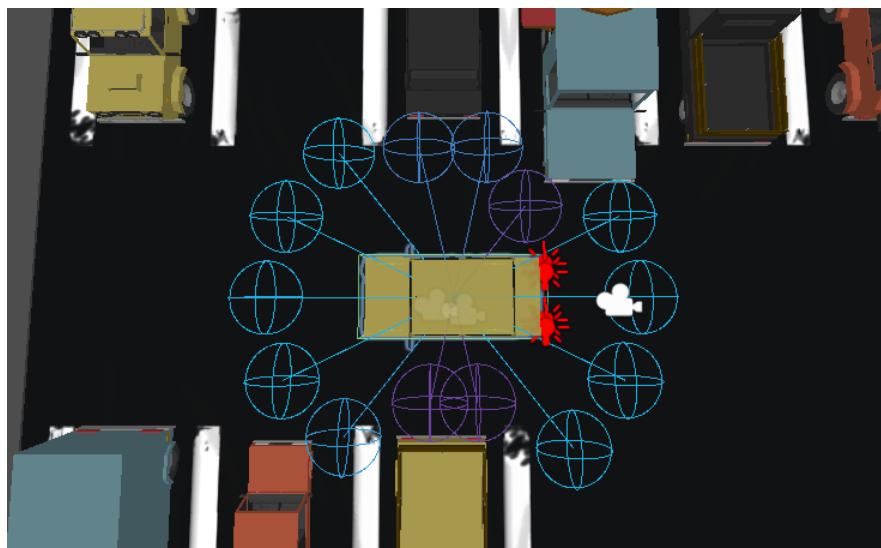


Figura 62 - Configurazione del sensore con raggi più piccoli

Avendo raggi molto più piccoli, rispetto alla sessione precedente, collide con l'ambiente circostante un quantitativo minore di essi.

#### Comportamento dell'agente ed eventuali problematiche

L'auto ha dimostrato maggiore libertà nei movimenti, ma i raggi piccoli hanno portato ad una grande problematica: ora l'auto non vede più la destinazione e cammina nuovamente a vuoto.

Siccome i raggi lunghi collidono con troppi ostacoli e i raggi corti non collidono con la destinazione, sembra non esserci alcuna soluzione se non una semplificazione della scena.

### 5.3.4 Parking-FS4-X

Parking-FS4, sessione *FSk* composta da 4 test.

#### Panoramica e idea

Dato il fallimento della seconda configurazione, si è deciso di caratterizzare il sensore RayPerception tornando alla prima configurazione, quindi con i raggi lunghi. L'idea, ora, consiste nel semplificare la scena del parcheggio rimuovendo qualche auto statica, in modo tale che l'auto osservi e raggiunga con più semplicità il posto auto libero. Inoltre, essendoci meno ostacoli nell'ambiente circostante, l'auto dovrebbe preoccuparsi meno riguardo la collisione con gli ostacoli.

Per risolvere il problema riguardante il fatto che l'auto non collidesse con il posto auto nonostante lo vedesse (ricordiamo che finché l'auto agente non tocca il posto auto, non saprà mai che assegna una ricompensa positiva), è stato necessario implementare il sottosistema di ricompense intelligente basato sulla distanza, mostrato nel capitolo 4.3.2 - *Sottosistema di ricompensa distanziale*, in modo da farle seguire la direzione corretta che l'avrebbe portata alla destinazione.

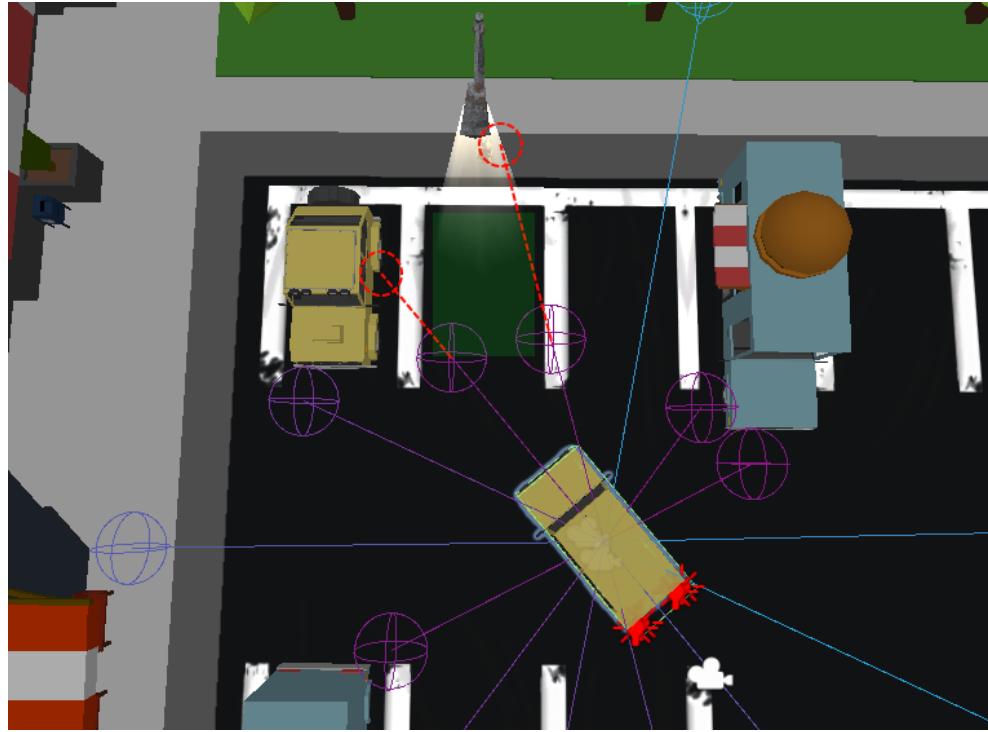


Figura 63 - Scena semplificata con meno auto statiche nel parcheggio

#### Comportamento dell'agente ed eventuali problematiche

Questa sessione è stata la prima a dare ottimi risultati riguardo la guida automatica dell'agente. Il fatto che l'agente osservi meno ostacoli implica il fatto che si senta più libero di muoversi, di provare nuove azioni e nuove manovre. L'auto, bene o male, riesce anche a fare qualche parcheggio in retromarcia, entrando anche abbastanza bene nel parcheggio. Un grosso problema, però, limita questa sessione: l'automobile agente è dotata di un sensore di rilevamento oggetti generico, quindi

tramite questo sensore riesce a vedere qualsiasi oggetto. Il problema consiste nel fatto che osservando un oggetto posto davanti a sé, il sensore non riesce a intravedere eventuali altri oggetti posti dietro quello osservato. Ciò causa non pochi problemi siccome l'auto, da determinate angolazioni, non riesce ad intravedere determinati ostacoli e dà per scontato che la strada sia libera, quindi che possa muoversi liberamente verso quella direzione.



*Figura 64 - La visuale viene limitata dall'osservazione del posto auto, quindi auto statica e lampioni non vengono osservati*

### 5.3 Sessioni LS<sub>k</sub>

Le sessioni *LS<sub>k</sub>* trattano il training in scene semplificate, quindi l'automobile agente parte da test con il parcheggio totalmente libero fino ad arrivare al parcheggio totalmente pieno. Ciò semplifica l'apprendimento dell'agente riguardo la guida autonoma siccome non dovrà preoccuparsi di ostacoli ed errori circa la sua posizione nel posto auto libero. La novità delle sessioni *LS<sub>k</sub>* riguarda l'utilizzo di ben due sensori RayPerception, i quali permettono all'automobile agente di visualizzare eventuali ostacoli posti vicino al posto auto di destinazione.

Inoltre, l'auto è caratterizzata da entrambi i sistemi di ricompense intelligenti mostrati nei capitoli 4.3.2 - *Sottosistema di ricompensa distanziale* e 4.3.3 - *Sottosistema di ricompensa direzionale*. L'introduzione del sottosistema di ricompensa direzionale

comporta grandi migliorie circa l'apprendimento delle sterzate: l'assegnamento di ricompense basato sull'angolo della sterzata confrontato all'angolo verso cui si trova la destinazione permette all'automobile agente di comprendere se il modo con cui sterza è corretto o meno.

Inoltre, nelle sessioni *LSk* l'area di gioco è definitivamente limitata al parcheggio, il quale ingresso è stato chiuso con un blocco invisibile.

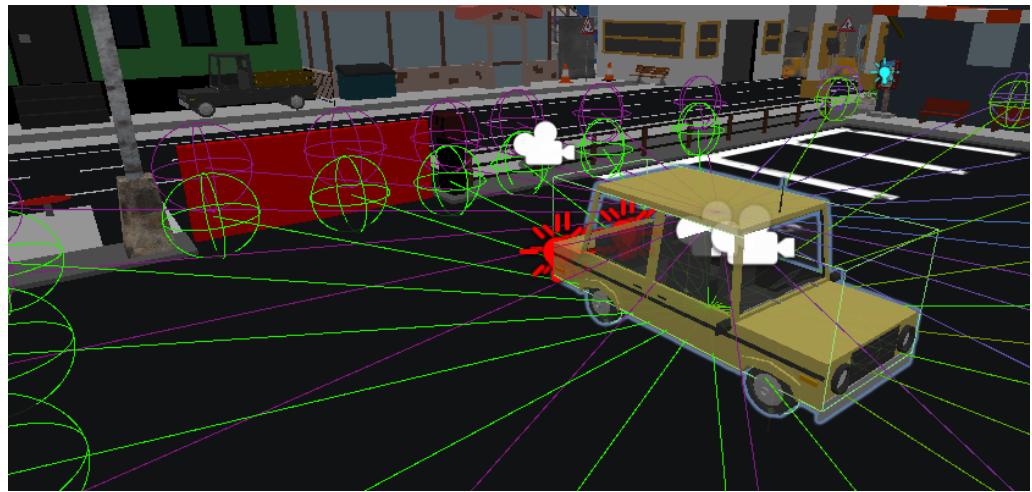


Figura 65 - Il parcheggio è ora limitato da un blocco *OutOfMap*

#### 5.4.1 Parking-LS1-X

Parking-LS1, sessione *LSk* composta da 4 test.

##### Panoramica e idea

L'area di gioco è limitata al solo parcheggio, partendo dall'averlo senza ostacoli fino ad avere un unico posto libero. L'idea di incrementare il numero di ostacoli man mano che l'auto impara è dovuta dal fatto che, con un parcheggio pieno, l'auto agente trova molte difficoltà nel migliorare la propria guida automatica.

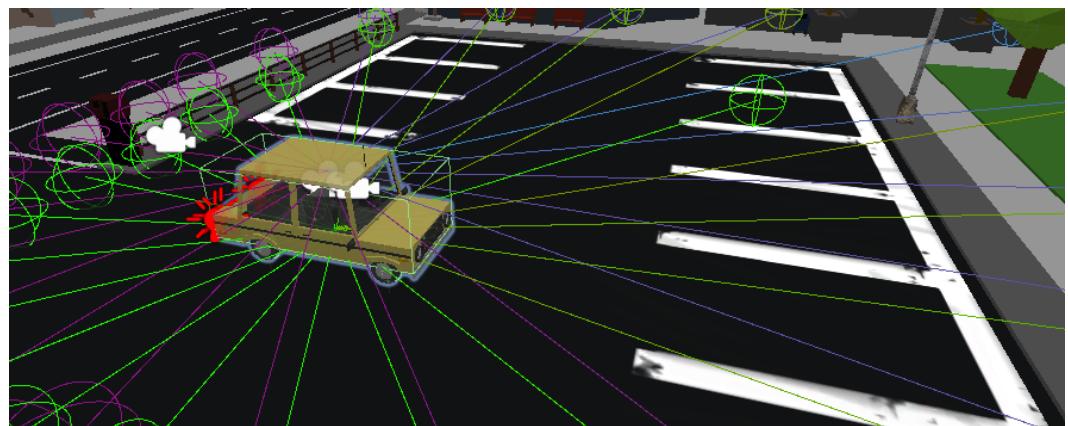
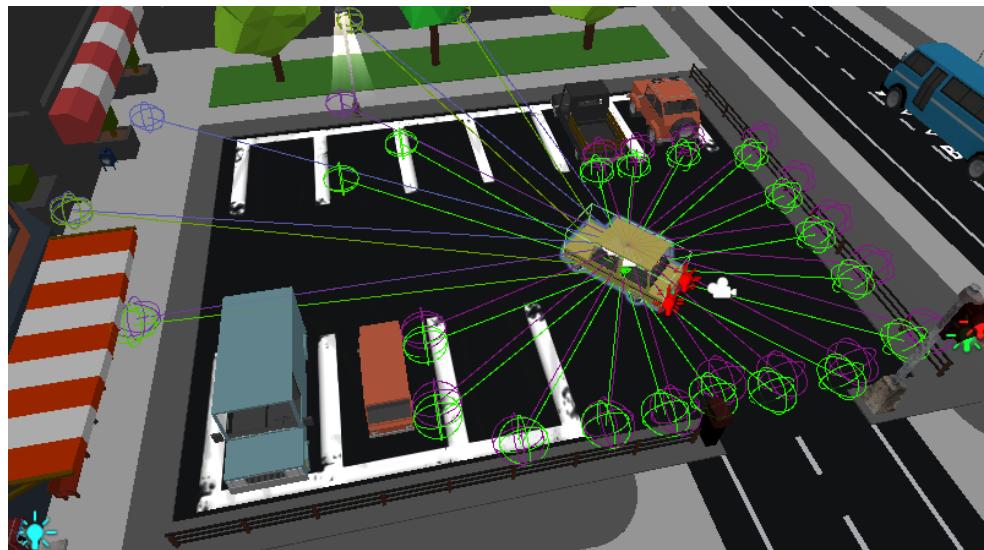


Figura 66 - Parcheggio semplificato

Un ragionamento del genere è molto più logico, siccome l'auto agente impara proprio come una persona alla guida per la prima volta: partire da uno scenario complesso complica le manovre e i semplici movimenti, quindi è bene semplificare la scena rimuovendo qualche ostacolo per poi riaggiungerlo quando si hanno risultati positivi riguardo l'apprendimento.

### **Comportamento dell'agente ed eventuali problematiche**

Il primo test della sessione ha permesso all'auto di imparare a camminare, sterzare e parcheggiare basilarmente nel parcheggio vuoto. Nel secondo test della sessione sono state aggiunte alcune auto statiche in posti auto che non influenzassero pesantemente il comportamento dell'auto agente, quindi bene o male il comportamento dell'auto agente è rimasto invariato.



*Figura 67 - Parcheggio parzialmente pieno*

Nei successivi due test 3-4 sono state aggiunte tutte le auto statiche nel parcheggio, quindi l'unico posto libero è l'effettiva destinazione. Nel complesso, l'auto riesce a completare la maggior parte dei parcheggi senza troppi problemi, ma in alcune manovre tende a bloccarsi tenendo una metà di essa all'interno e l'altra metà all'esterno del posto auto. Si sospetta sia dovuto dal fatto che l'auto agente non abbia ben imparato l'uso della retromarcia, quindi nel quarto test sono stati utilizzati molteplici spawn orientati alla retromarcia: l'auto rimane bloccata sugli spawn, non capendo come muoversi.

Da ciò consegue il fatto che l'auto non ha ben imparato ad utilizzare la retromarcia ed è necessario farle imparare nuovamente i movimenti più basilari.



Figura 68 - L'auto rimane bloccata vicino al parcheggio

## Risultati

Nel complesso, l'auto ha comunque assunto un buon comportamento e riesce a guidare relativamente bene, eccetto per la retromarcia. Seguono quattro grafici di TensorBoard relativi ai quattro test che compongono la sessione.

È bene ricordare che più il grafico è stabile e migliori sono i risultati, come illustrato nel capitolo 4.3.1 - *Sistema di ricompense*.



Figura 69 - Parking-LS1-1: learning guida automatica

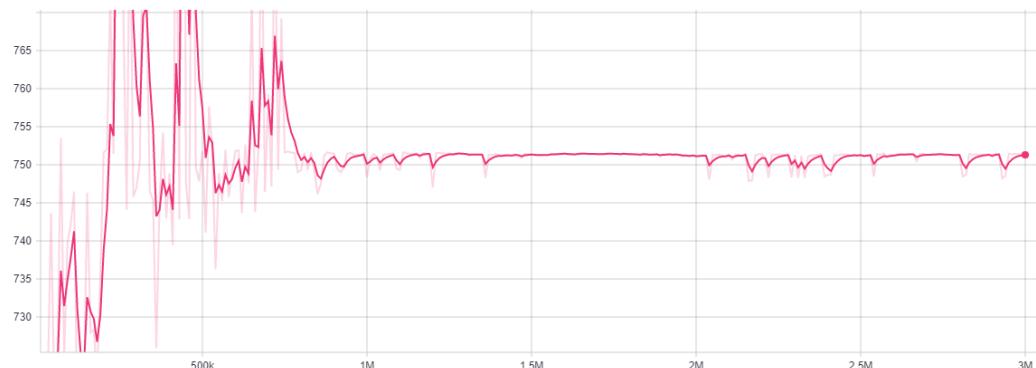
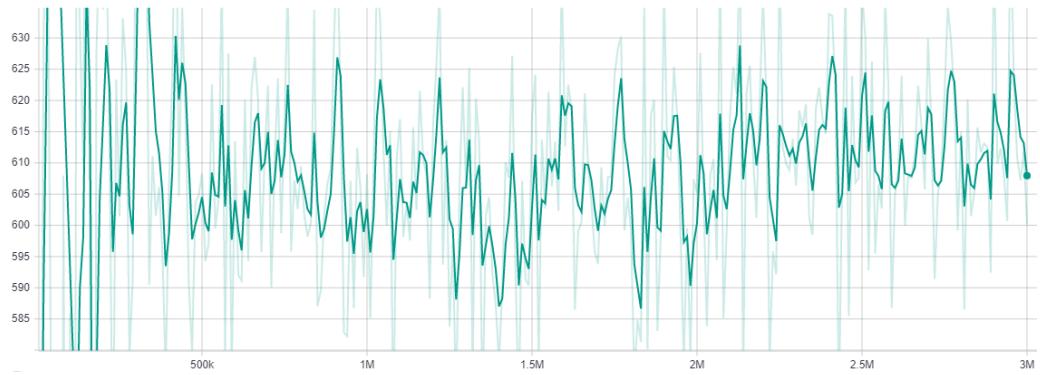
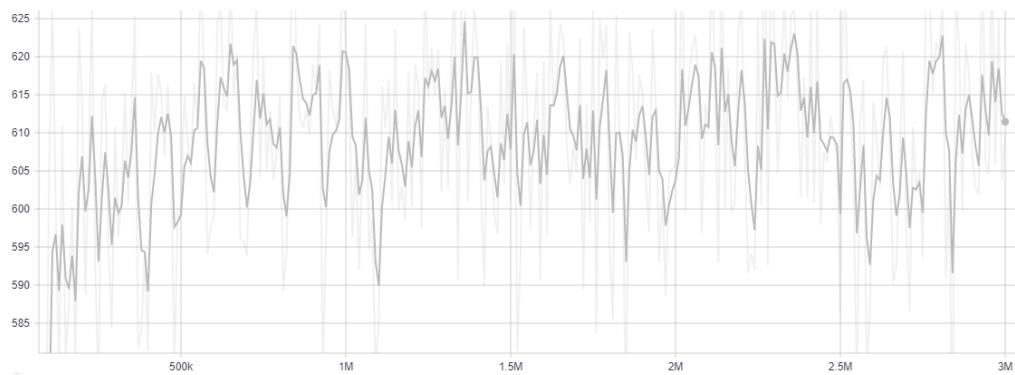


Figura 70 - Parking-LS1-2: learning guida automatica con qualche auto nel parcheggio

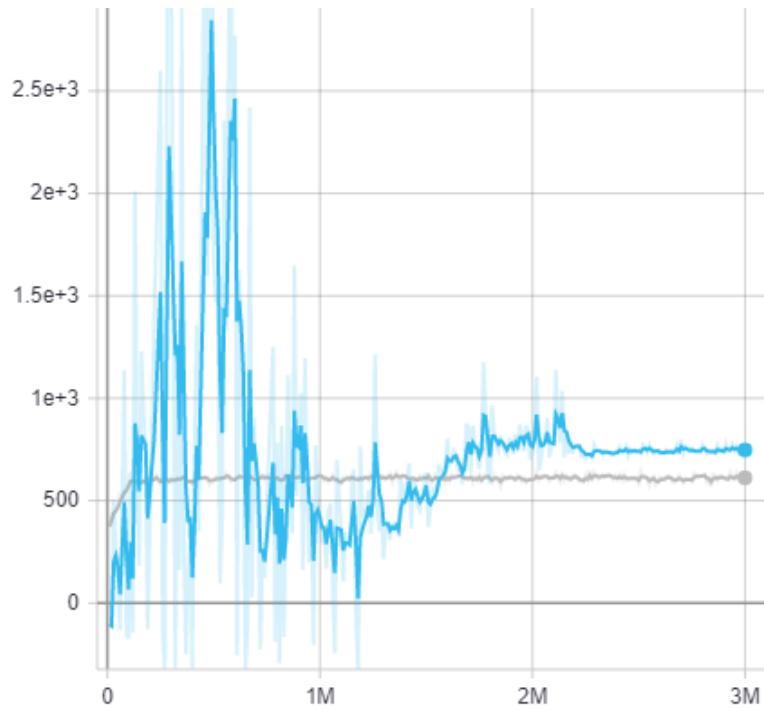


*Figura 71 - Parking-LS1-3: learning posizionamento automatico nel parcheggio pieno*



*Figura 72 - Parking-LS1-4: learning posizionamento automatico con retromarcia*

Nelle immagini può sembrare che i test 3 e 4 abbiano grossi picchi, ma non è così: è solo una questione di zoom sui grafici. Per misurare i picchi è necessario riferirsi ai valori posti a sinistra.



*Figura 73 - Test 1 e 4 messi a paragone*

I miglioramenti ci sono e, nonostante i problemi con la retromarcia, come osservabile dai grafici, il comportamento tende a non avere grossi picchi a partire dal terzo test: mentre nei primi due test troviamo variazioni di media di 500+ punti, nei successivi due test troviamo variazioni fino a 50 punti circa, il che è decisamente un buon risultato.

### 5.4.2 Parking-LS2-X

Parking-LS2, sessione *LSk* composta da 13 test.

#### Panoramica e idea

Nella sessione precedente l'auto agente non riusciva ad effettuare parcheggi e manovre in retromarcia, quindi questa sessione nasce per allenarla sin da subito con la retromarcia seguendo, in generale, gli stessi schemi della precedente sessione.

#### Comportamento dell'agente ed eventuali problematiche

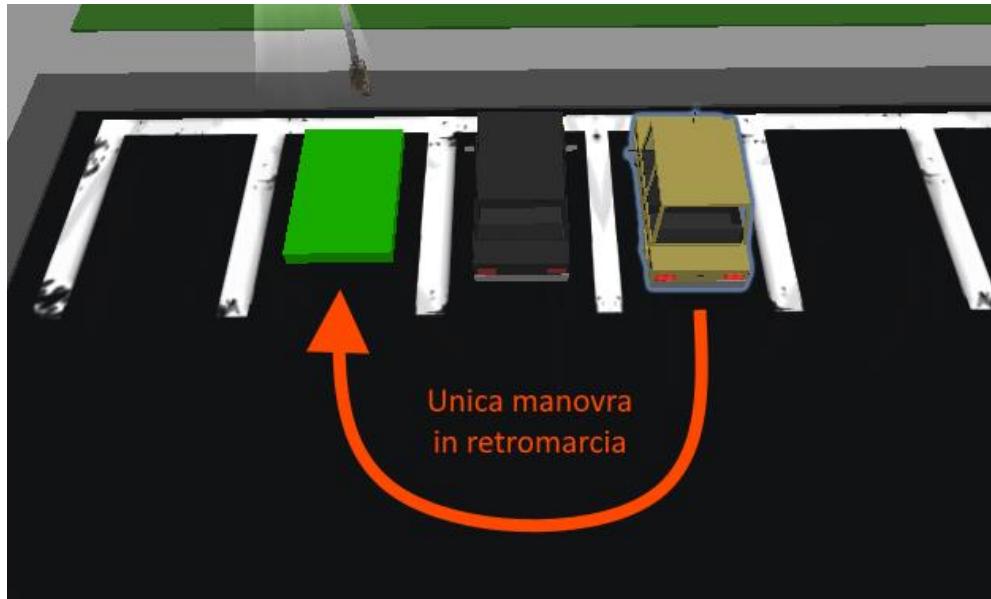
I primi tre test (1-2-3) sono dedicati ai movimenti dell'auto agente nel parcheggio vuoto: deve imparare ad effettuare parcheggi da quante più posizioni di spawn che favoriscono allo stesso modo la marcia frontale e la retromarcia, quindi a differenza della sessione precedente sono state aggiunte molte posizioni iniziali che favoriscono l'uso della retromarcia.



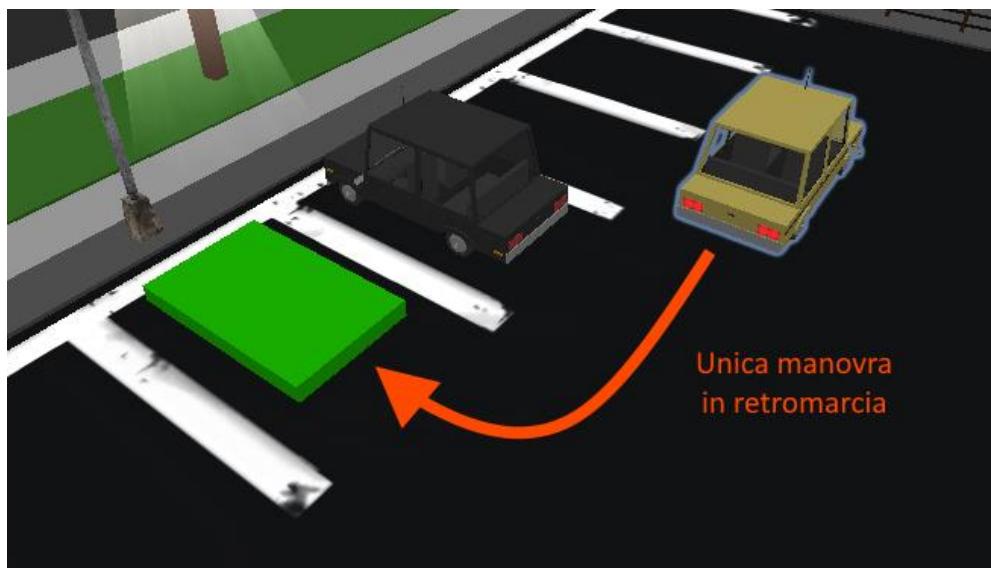
Figura 74 - Posizione di spawn favorevole alla retromarcia

Nonostante ciò, durante i test si nota che, in alcuni spawn favorevoli alla retromarcia, l'auto preferisce comunque effettuare parcheggi frontali anziché in retromarcia: è disposta a fare strane manovre al fine di raddrizzarsi e posizionarsi frontalmente.

I successivi tre test (4-5-6) sono dedicati all'apprendimento dell'auto nel parcheggio con un'unica auto statica all'interno, posta al fianco del posto libero di destinazione: questo per forzare il parcheggio in retromarcia da determinati punti di spawn, in modo da costringere l'auto ad imparare determinate manovre.



*Figura 75 - Posizione di facilitazione per la retromarcia*



*Figura 76 - Altra posizione di facilitazione per la retromarcia*

I prossimi quattro test (7-8-9-10) sono dedicati all'apprendimento dell'auto sul parcheggio con ben due auto statiche poste all'interno, entrambe posizioionate lateralmente al posto libero, in modo tale che l'auto agente impari sin da subito ad effettuare manovre più precise.

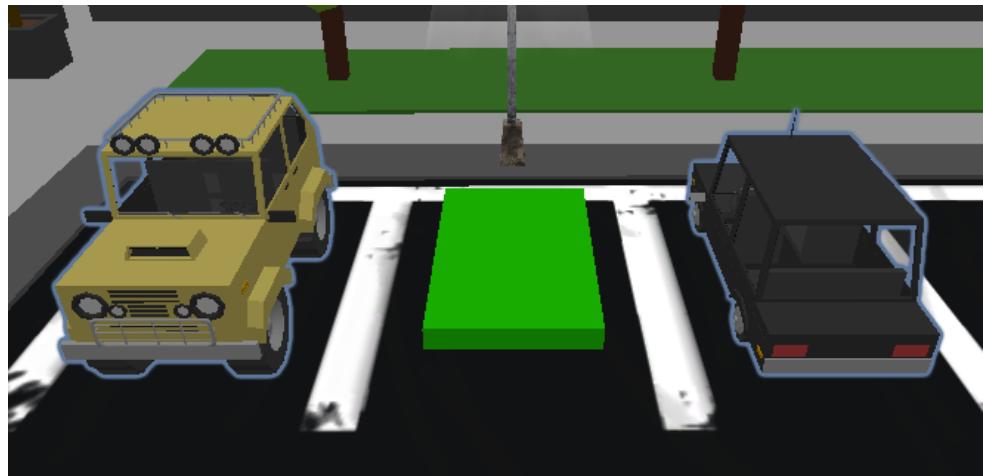


Figura 77 - Auto statiche laterali al posto libero

Essendo occupati entrambi i posti laterali a quello libero, l'auto agente ha meno spazio a disposizione in cui effettuare manovre e richiede molti più step per imparare ad essere precisa nelle manovre.

Negli ultimi tre test (11-12-13) sono state aggiunte alcune auto statiche nel parcheggio, fornendo ancora all'auto agente spazio libero in cui giostrarsi con le manovre. Ciò ha migliorato l'attenzione prestata nelle manovre verso le auto ostacolo, sia frontali che in retromarcia.

## Risultati

L'auto effettua ottime manovre prestando attenzione verso gli ostacoli e molta precisione nelle sterzate. Il risultato è più che soddisfacente, ma la sessione non è stata conclusa: manca l'apprendimento nel parcheggio pieno di auto statiche.

La sessione non è stata conclusa perché il box indicante il posto libero in cui parcheggiare è troppo largo e l'auto può permettersi di parcheggiare relativamente male, pensando di aver fatto un perfetto parcheggio.

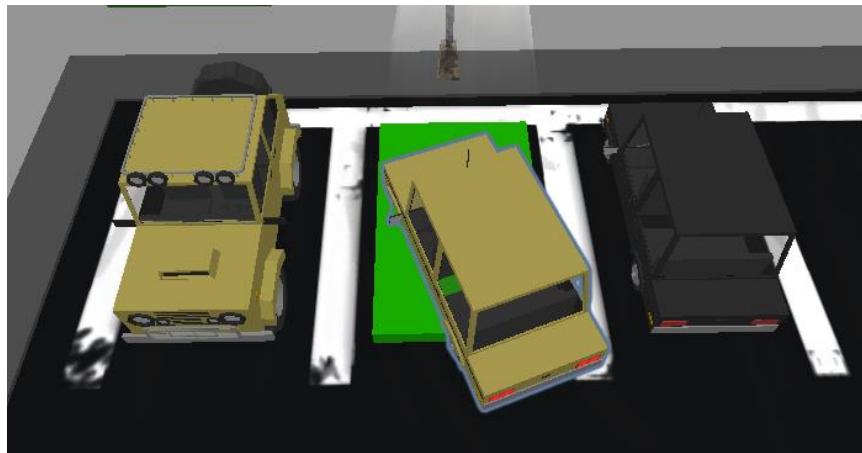


Figura 78 - Box posto libero troppo grande: l'auto può parcheggiare male

*Nota: tutte le immagini finora viste raffigurano il box del posto libero di grandezza corretta. L'immagine appena riportata mostra la grandezza del box del posto libero originaria, quindi quella scorretta. La modifica della sua grandezza avviene, in realtà, nella prossima sessione di learning.*

Seguono alcuni grafici di TensorBoard relativi ai test che compongono la sessione. Alcuni test verranno mostrati insieme in un unico grafico, al fine di non riportare ben 13 grafici, uno per ogni test.

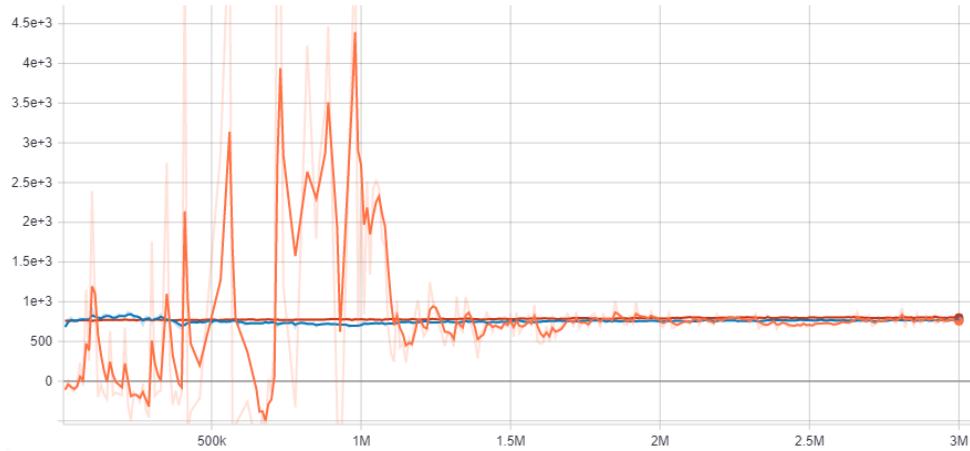


Figura 79 - Parking-LS2-{1-2-3}: learning guida automatica  
Test 1: arancione - Test 2: blu - Test 3: rosso

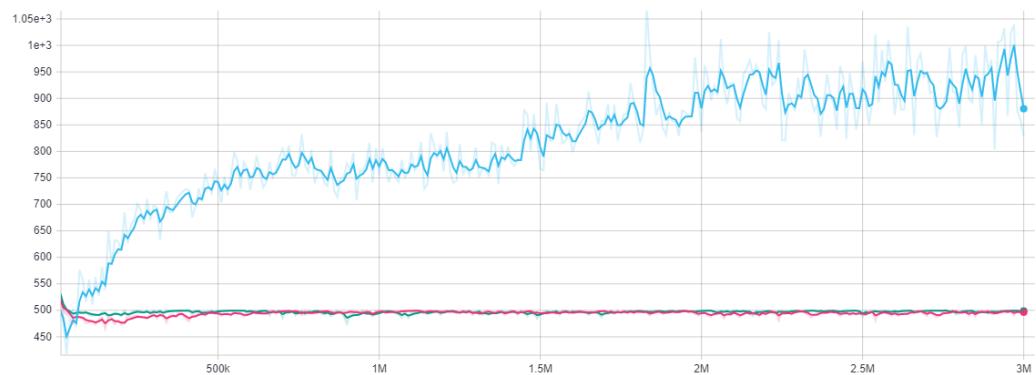


Figura 80 - Parking-LS2-{4-5-6}: learning guida automatica con una sola auto statica  
Test 4: celeste - Test 5:magenta - Test 6: verde acqua

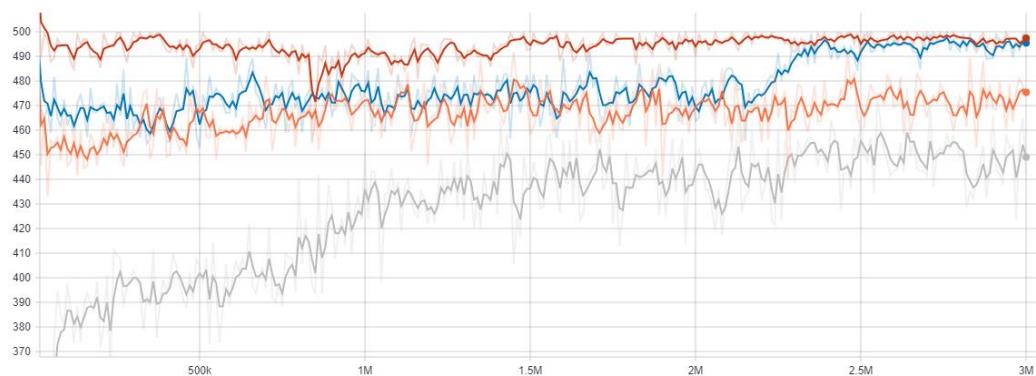


Figura 81 - Parking-LS2-{7-8-9-10}: learning guida automatica con due auto statiche  
Test 7: grigio - Test 8: arancione - Test 9: blu - Test 10: rosso

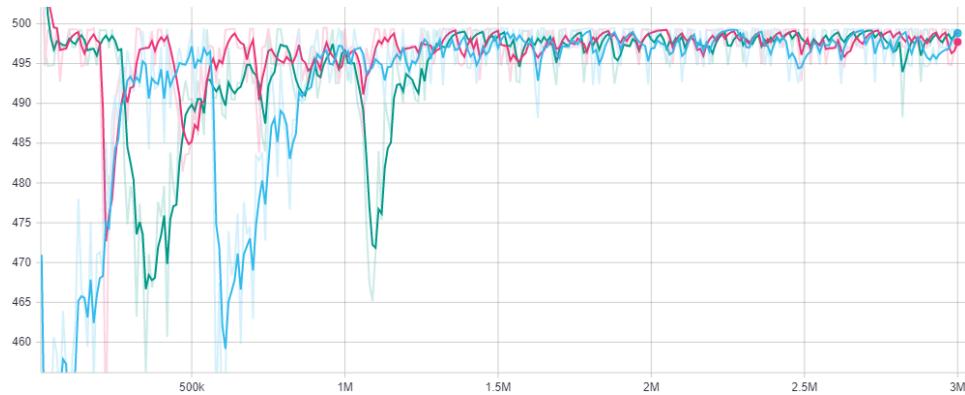


Figura 82 - Parking-LS2-{11-12-13}: learning guida automatica con parcheggio parzialmente pieno  
Test 4: celeste - Test 5:magenta - Test 6: verde acqua

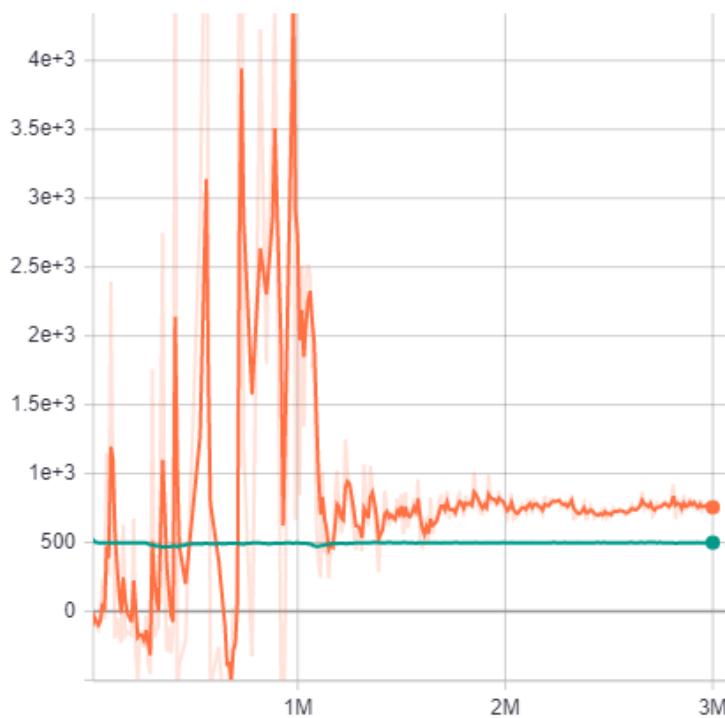


Figura 83 - Test 1 e 13 messi a paragone

Quest'ultima immagine mostra benissimo i risultati avuti: nel primo test l'auto non sapeva nemmeno camminare; nell'ultimo test l'auto parcheggia benissimo.

La differenza dei picchi tra i due test messi a paragone è decisamente evidente: mentre nel primo test si avevano picchi di 5000+ punti sulla media, nell'ultimo test si hanno picchi di 5-15 punti circa. Il comportamento dell'auto agente alla conclusione della seconda sessione *LSk* è più che stabile e da ritenersi più che ottimo.

L'unico aspetto da migliorare riguarda la grandezza del box del posto auto libero, il quale sarà più piccolo e richiederà maggiore precisione all'auto agente.

### 5.4.3 Parking-LS3-X

Parking-LS3, sessione  $LSk$  composta da 12 test.

#### Panoramica e idea

La terza sessione  $LSk$  tende ad essere una sessione di miglioramento per la seconda, siccome riduce la grandezza del box del posto libero, al fine di avere un posizionamento da parte dell'agente molto più preciso e curato.

Prima di vedere il comportamento dell'agente e di analizzare i diversi test effettuati, è bene sapere una cosa: ogni sessione finora vista tende a ripartire da zero, nel senso che l'auto torna a sperimentare ed imparare senza sapere nulla. Avendo dalla seconda sessione  $LSk$  una guida automatica più che buona, la terza sessione fa uso della NeuralNetwork generata dai test Parking-LS2-{1-2-3}: ciò significa che l'agente, in questa sessione, non imparerà da zero, bensì partirà dall'aver già imparato i movimenti e le manovre fondamentali.

#### Comportamento dell'agente ed eventuali problematiche

Tale sessione parte da una Neural Network già sviluppata dai primi tre test della precedente sessione, quindi il primo test di questa attuale sessione allenerà i movimenti assunti precedentemente con un box posto auto più piccolo in un parcheggio senza auto statiche.

I successivi test aggiungono, man mano, tutte le auto statiche nel parcheggio, fino a rimanere un unico posto libero in cui parcheggiare.

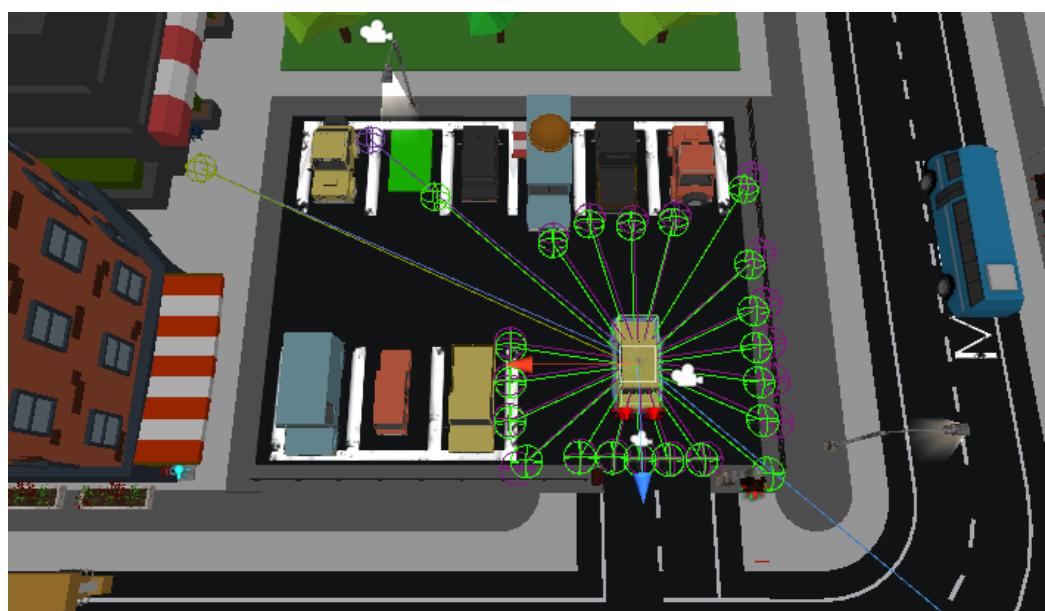


Figura 84 - Parcheggio conclusivo

L'auto, con i test della terza sessione, ha imparato a parcheggiare con successo, effettuando manovre con assoluta precisione e accuratezza.

## Risultati

I risultati riscontrati in questa sessione sono leggermente meno bilanciati rispetto a quelli della sessione precedente, ma l'auto ha assunto un comportamento più che buono al punto da rendere trascurabile la leggera differenza.

Vediamo, piuttosto, le oscillazioni messe a confronto tra gli ultimi grafici delle due sessioni.



Figura 85 - Parking-LS2-13: learning guida automatica con parcheggio parzialmente pieno

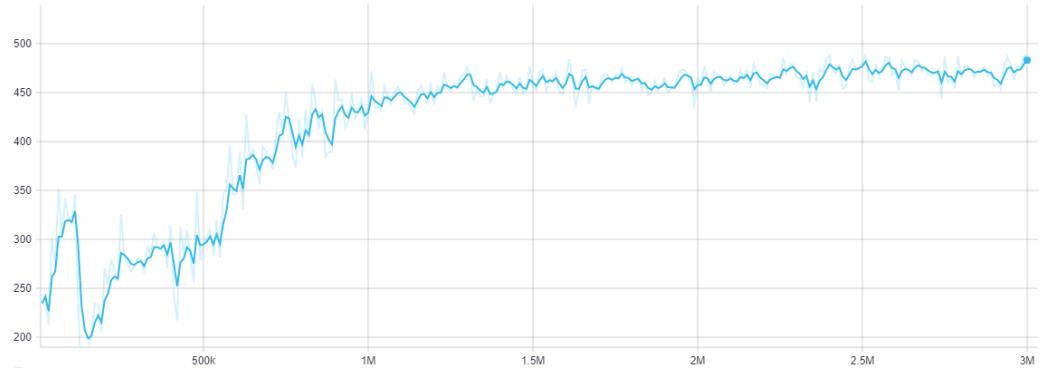


Figura 86 - Parking-LS3-12: learning guida automatica con parcheggio pieno

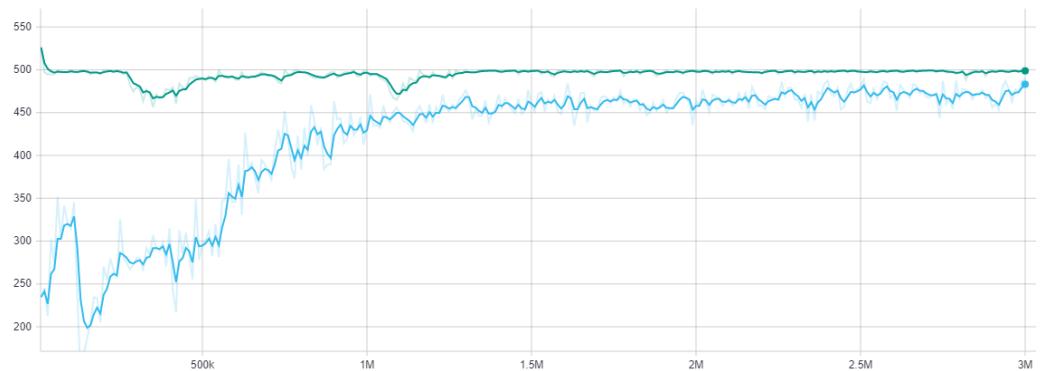


Figura 87 - Test LS2-13 e LS3-12 messi a paragone

Come anticipato, la differenza dei picchi tra i due test messi a paragone è minima: tenendo conto del fatto che, a differenza di quest'ultimo test, quello della sessione precedente si basi su un parcheggio non completo e con un box posto auto più largo, i picchi mostrati sono da ritenersi un ottimo traguardo, siccome in un parcheggio completo e con un box posto auto più stretto il margine di errore è molto più alto.

Se nel test Parking-LS2-13 si avevano picchi di circa 5-15 punti in media, nel test Parking-LS3-12 si hanno picchi di circa 10-20 punti in media e 40 circa al massimo, il che è comunque buono.

Concludendo, quindi, l'auto parcheggia in maniera precisa e realistica nel posto auto, prestando attenzione all'ambiente circostante ed evitando collisione con qualsiasi ostacolo durante qualsiasi manovra.

# Capitolo 6

# Prototipo e sviluppi futuri

## 6.1 Build

Come precedentemente ribadito, il progetto AI Car Kineton è composto di due scene indipendenti, le quali trattano reciprocamente due problematiche molto differenti tra loro, quali sono il parcheggio automatizzato e il rilevamento automatizzato di pedoni su strada.

La build del progetto, cioè il prototipo, permette all'utente di avere una visione completa di ciò che lo compone tramite un piccolo menù, il quale permette di visualizzare i risultati finali dei training su entrambe le scene, quindi di vedere come l'automobile compie scelte intelligenti in entrambe le situazioni.

Altra funzionalità messa a disposizione per l'utente è la guida manuale in una scena completa in cui sono compresi il parcheggio e molteplici attraversamenti pedonali. In questa scena, l'automobile non sarà dotata di alcun sensore e sarà l'utente ad averne il pieno controllo tramite comandi da tastiera.

### 6.1.1 Funzionamento del menu

Il menu segue una struttura semplice e lineare e permette all'utente di visualizzare le scene di parcheggio automatizzato e di rilevamento automatico dei pedoni e di provare l'auto manualmente in una scena dedicata.

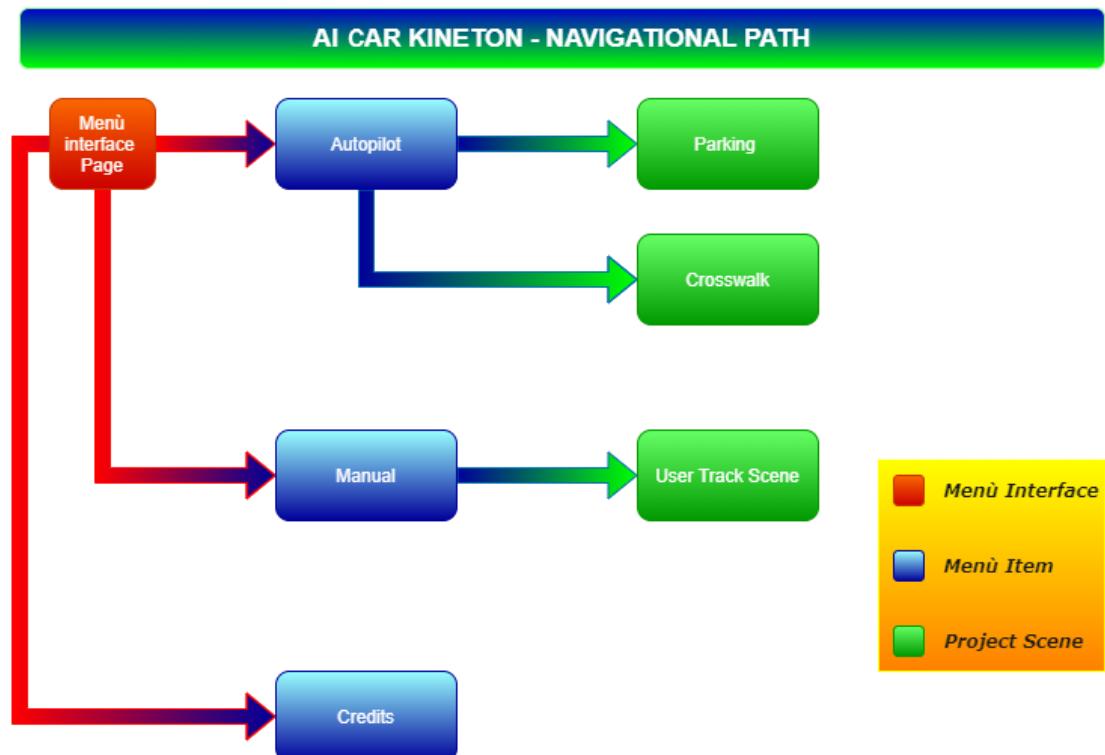


Figura 88 - Path di navigazione del menu

Il menu è molto dinamico siccome fa utilizzo di piccole e simpatiche animazioni che rendono il tutto molto più gradevole. Le animazioni sono state realizzate giocando sulla sottolineatura del testo, sulla trasparenza e sulle posizioni, quindi su attributi primitivi del testo; il tutto viene gestito tramite due strutture create internamente al menu, quali sono il DecoButton e la DecoAnimation, i quali, rispettivamente, facilitano la gestione dei pulsanti e delle animazioni.

La logica del menu, utilizzando queste due strutture, risulta abbastanza semplice:

- Se l'utente naviga in una sezione del menu, il DecoButton sottolinea il testo su cui l'utente si trova;
- Se l'utente clicca un pulsante, vengono create delle DecoAnimation che permettono la visualizzazione dei pulsanti della nuova sezione del menu, facendoli comparire rimuovendo la trasparenza da essi;

- Se l'utente torna nella precedente sezione, vengono create delle DecoAnimation che nascondano i pulsanti della sezione da rimuovere, applicando la trasparenza ad essi.

```
class DecoButton {
    // Internal attributes
    private string text;
    private TMP_Text guiText;
    // Constructor
    public DecoButton(string text, TMP_Text guiText) ...
    /*
        Functions
    */
    public string getText() ...
    public TMP_Text getGUI() ...
    /*
        updateGUI(bool underlined)
            Manages the underlining of the text
    */
    public void updateGUI(bool underlined = false) ...
    /*
        changeAlpha(float alpha)
            Apply a new transparency to the text
    */
    public void changeAlpha(float alpha) ...
    public float getAlpha() ...
    /*
        changeAlphaOfIcon(float alpha)
            Apply a new transparency to the text's icon
    */
    public void changeAlphaOfIcon(float alpha) ...
    public float getAlphaOfIcon() ...
}
```

Figura 89 - Struttura DecoButton

```
class DecoAnimation {
    // Internal attributes
    private DecoButton button;
    private bool start;
    public AnimationType type;
    // Constructor
    public DecoAnimation(DecoButton button, bool start, AnimationType type) ...
    /*
        Functions
    */
    public DecoButton getButton() ...
    public bool isStart() ...
    public void setStatus(bool visible) ...
}
```

Figura 90 - Struttura DecoAnimation

```
foreach (DecoButton b in modeButtons) {
    b.changeAlpha(0f);
    b.changeAlphaOfIcon(0f);
    b.updateGUI();
    toAnimate.Add(new DecoAnimation(b, false, AnimationType.FADE));
    backupToAnimate.Add(new DecoAnimation(b, false, AnimationType.FADE));
}
```

Figura 91 - Esempio di utilizzo di DecoButton e DecoAnimation

La struttura DecoAnimation supporta anche altre animazioni oltre alla comparsa e la scomparsa del testo, come lo spostamento verso l'alto e verso il basso di quest'ultimo.

La tipologia di animazione da eseguire viene scelta alla creazione di un’istanza della struttura.



*Figura 92 - Menu interattivo*

## 6.2 Sviluppi futuri

Il progetto di tirocinio AI Car Kineton sì è fin da subito distinto per la sua ottima scalabilità, infatti anche durante l’intero iter progettuale, molte sono state le aggiunte e complicazioni apportate alle simulazioni, basti pensare il progressivo ampliamento di veicoli, con il quale l’agente si è misurato nella scena di parcheggio automatico, oppure all’aggiunta di più pedoni o il doppio senso di marcia per la scena di attraversamento pedonale. Oltre tutto, il compito fondamentale di un sistema di sensoristica ADAS, è proprio quello di fornire un supporto pronto a rispondere alle più svariate e complesse eventualità riscontrabili in una quotidiana esperienza di guida urbana o extraurbana.

In quest’ottica è molto semplice ragionare in termini di sviluppi progettuali futuri per il progetto implementato, basti pensare alle innumerevoli possibilità di dinamizzazione che ancora possono essere implementate su entrambe le simulazioni:

- Per la scena di parcheggio, ad esempio, è possibile considerare la possibilità di effettuare altre manovre come ad esempio quelle utili alla realizzazione di un parcheggio in colonna. Altra possibilità consiste nel rendere l’autovettura ancora più libera nel suo percorso, avviando la simulazione su strada invece

che dall'interno del parcheggio stesso, addirittura rendendo, magari, la locazione del posto libero randomica tra una serie di posti auto precedentemente definiti. Ciò rende la ricerca del posto auto più dinamica;

- Per la scena di attraversamento pedonale, invece, un'ottima possibilità di espansione è ancora presente nell'interazione tra pedoni e agente, ad esempio rendendo dinamiche e casuali le coordinate spaziali o temporali di attraversamento di ogni singolo pedone, rendendo l'agente ancora più versatile e pronto a rispondere correttamente alla casualità con cui un pedone può trovarsi su strada in un contesto reale.

Inoltre, è utile precisare che AI Car Kineton è stato concepito e progettato al fine di rispondere alle più svariate esigenze stradali, in cui un'autovettura è dotata di supporti basati sull'intelligenza artificiale.

Non a caso, come visibile dalla scena adibita alla guida manuale, è stato predisposto anche un incrocio stradale, nel quale sarebbe davvero molto interessante provare a specializzare i sensori di cui l'agente già è dotato, ad esempio implementando un sistema per il riconoscimento della segnaletica stradale, un meccanismo di precedenze tra varie autovetture, oppure una simulazione adibita alla circolazione rotatoria.

Infine, è importante dare uno sguardo alla possibilità di esportare e adattare le Neural Network prodotte in un contesto di simulazione differente rispetto a quello utilizzato: infatti le Neural Network prodotte da ML Agents possono essere facilmente esportate e adattate a piccoli prototipi di veicoli reali. Insomma, sarebbe addirittura possibile considerare l'ipotesi di uscire dall'ambito di simulazione e di passare alla progettazione di un piccolo progetto di automazione in ambito robotico, con l'ausilio di sensoristiche di vario tipo atte a simulare sempre di più il comportamento di un reale sensore ADAS da installare su un'autovettura.

Nonostante le proposte riportate, oggigiorno la sicurezza stradale necessita davvero tanto di supporti di sensoristica ADAS per varie realtà che un conducente può affrontare e dell'applicazione di Machine Learning e Intelligenza Artificiale, le quali garantiscono un ottimo grado di affidabilità. A tale scopo, AI Car Kineton lascia ampli spazi di libertà a chiunque voglia cimentarsi in questo enorme mondo, che è la sicurezza urbana, dando la possibilità di integrare, tramite una vasta gamma di scelte, le più svariate situazioni che necessitano di automazione e sicurezza nel complesso mondo dell'automotive.



# Bibliografia

- [1] Road traffic injuries. Available: <https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>
- [2] Tesla autopilot. Available: [https://www.tesla.com/it\\_IT/autopilot](https://www.tesla.com/it_IT/autopilot)
- [3] Adas, quali sono i sistemi di assistenza alla guida. Available:  
<https://www.newsauto.it/guide/adas-quali-sono-significato-2020-111281/>
- [4] How do self-driving cars work? Available: <https://robohub.org/how-do-self-driving-cars-work/>
- [5] Mathworks, *Machine Learning with MATLAB*. Chapter 1. Available:  
<https://www.mathworks.com/campaigns/offers/machine-learning-with-matlab.html?elqCampaignId=10588>
- [6] Machine Learning - Intelligenza Artificiale. Available:  
<https://www.intelligenzaartificiale.it/machine-learning/>
- [7] Neural Network. Available: <https://deepai.org/machine-learning-glossary-and-terms/neural-network#:~:text=An%20artificial%20neural%20network%20learning,output%2C%20usually%20in%20another%20form.>

- [8] Reti neurali: cosa sono e a cosa servono. Available:  
<https://www.ai4business.it/intelligenza-artificiale/deep-learning/reti-neurali/>
- [9] Machine Learning for Beginners: An Introduction to Neural Networks.  
Available: <https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9>
- [10] Using neural nets to recognize handwritten digits. Chapter “Perceptrons”.  
Available: <http://neuralnetworksanddeeplearning.com/chap1.html>
- [11] Ethem Alpaydin, *Introduction to Machine Learning*. Chapter 19. Available:  
[https://books.google.it/books?hl=it&lr=&id=tZnSDwAAQBAJ&oi=fnd&pg=PR7&dq=machine+learning&ots=F2\\_U7Xdrwe&sig=4qkOlog5qmHhKJmnKqJuJ2qZbFQ](https://books.google.it/books?hl=it&lr=&id=tZnSDwAAQBAJ&oi=fnd&pg=PR7&dq=machine+learning&ots=F2_U7Xdrwe&sig=4qkOlog5qmHhKJmnKqJuJ2qZbFQ)
- [12] Artificial Intelligence: what’s the difference between Deep Learning and Reinforcement Learning? Available:  
<https://www.forbes.com/sites/bernardmarr/2018/10/22/artificial-intelligence-whats-the-difference-between-deep-learning-and-reinforcement-learning/#4e086e58271e>
- [13] Game Engines: how do they work? Available: <https://unity3d.com/what-is-a-game-engine>
- [14] Alan Thorn, *Game Engine Design and Implementation*. Chapter 1.  
Available: [https://books.google.it/books?hl=it&lr=&id=cFnPvWK9-akC&oi=fnd&pg=PR3&dq=video+game+engine&ots=nkNIBoQHTW&sig=tZoxfAWzhXRRfkEV5Zr2XFzt8A&redir\\_esc=y#v=onepage&q&f=false](https://books.google.it/books?hl=it&lr=&id=cFnPvWK9-akC&oi=fnd&pg=PR3&dq=video+game+engine&ots=nkNIBoQHTW&sig=tZoxfAWzhXRRfkEV5Zr2XFzt8A&redir_esc=y#v=onepage&q&f=false)
- [15] Jared Halpern, *Developing 2D Games with Unity*. Chapter 1-2. Available:  
<https://www.amazon.it/Developing-Games-Unity-Independent-Programming-ebook/dp/B07FKFVFTML>
- [16] Sue Blackman, *Unity for Absolute Beginners*. Introduction, Chapter 1.  
Available: <https://www.amazon.it/Unity-Absolute-Beginners-English-Blackman-ebook/dp/B01HXKK5DG>
- [17] Coding in C# in Unity for beginners. Available:  
<https://unity3d.com/learning-c-sharp-in-unity-for-beginners>
- [18] Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D. (2020). Unity: A General

Platform for Intelligent Agents. *arXiv preprint arXiv:1809.02627*. <https://github.com/Unity-Technologies/ml-agents>.

[19] TensorFlow documentation. Available: <https://www.tensorflow.org/>