POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo di un framework per la gestione di animazioni e sintesi vocale di Empathic Embodied Conversational Agents

Relatore
prof. Andrea Giuseppe Bottino

Correlatori:
dott. Edoardo Battegazzorre
dott. Francesco Strada

Laureandi
Francesco Cali'
Pietro Pingitore

Anno accademico 2020-2021

Indice

Introduzione

Prefazione

L'intelligenza artificiale è uno dei topic più dibattuti all'interno della comunità scientifica. La ricerca sta andando verso una maggiore complessità di interazione e di utilizzo di agenti intelligenti soffermandosi principalmente sulla precisione dei task e sulla velocità di calcolo. Nella nostra ricerca, invece, si affronta la sfida di rendere l'intelligenza artificiale più simile possibile ad un essere umano. Partendo da una definizione basilare di un Empathic Embodied Conversational Agent basiamo la nostra ricerca nella definizione di animazioni consone all'essere umano e nella creazione automatica di frasi espresse con empatia.
L'empatia è un fattore poco trattato dalla comunità scientifica che, però, rende un'intelligenza artificiale più vicina all'utente che ci si interfaccerà. Nel caso in cui l'interazione sia uomo-macchina percepire l'agente considerato come un essere più simile all'essere umano renderà l'interazione più gradevole e meno arida; mentre nel caso in cui l'interazione sia passiva e, quindi, si basi solo sull'osservazione di uno o più agenti interagire fra loro o con un ambiente tridimensionale, la similarità con l'essere umano renderà la fruizione più interessante e partecipativa.
Gli ambiti che potrebbero giovare di un agente intelligente ed empatico sono svariati (beni culturali, medicina, training, ecc.) e per ogni ambito si potrebbero avere diverse applicazioni in base alle necessità. Per questo ventaglio di possibilità l'obiettivo fondamentale è quello di rendere l'utilizzo dell'ECA il più modulabile e customizzabile possibile, in modo che ogni utente possa personalizzare l'agente a proprio piacimento.

Struttura della ricerca

La struttura di ricerca è basata su più fasi:

- Ricerca, utilizzo e modifica di un framework di animazione, in particolare il "Motion Matching Animation System" poiché più funzionale alla definizione ed al blending di animazioni in runtime rispetto all'animator di Unity;
- Strutturazione del codice in C Sharp per la definizione di azioni modulabili e creabili ex novo;
- Ricerca, utilizzo e modifica di una rete neurale Tacotron per l'implementazione delle funzioni di Text-To-Speech.

Capitolo 1

Stato dell'Arte

1.1 Intelligenza Artificiale

1.1.1 Introduzione

L'intelligenza artificiale è il ramo dell'informatica che riesce ad unire le caratteristiche di una macchina a quelle dell'essere umano. Quindi, con il termine intelligenza, si va oltre il concetto puramente calcolistico ma si definiscono tutte le "intelligenze" definite da Gardner [5]:

- Intelligenza logico-matematica
- Intelligenza linguistica
- Intelligenza spaziale
- Intelligenza musicale
- Intelligenza cinestetica o procedurale
- Intelligenza interpersonale

Figura 1.1: Intelligenze multiple di Gardner

Un sistema intelligente è in grado di riprodurre una o più di queste intelligenze. Sono tre i cardini fondamentali del comportamento umano che sono alla base della programmazione di un'IA: avere una conoscenza non sterile, una coscienza che faccia prendere decisioni non solo secondo logica e l'abilità di risolvere i problemi in maniera differente.

1.1.2 Storia

La nascita del concetto di intelligenza artificiale risale al 1956: durante la conferenza di Dartmouth[13] venne definito il concetto di **Sistema Intelligente** capace di effettuare ragionamenti logici legati alla matematica.
Il periodo successivo a questo convegno fu di grande fermento intellettuale che portò alla creazione di diversi software in grado di risolvere elaborazioni matematiche complesse, ma anche alla scoperta delle limitazioni dell'IA: non essendo un vero e proprio cervello mancava dell'intuito e della logica propriamente umane. La conseguenza delle limitazioni fu una progressiva diminuzione dei finanziamenti e dell'interesse di ricerca nei confronti dell'IA.
Il nuovo impulso alla ricerca per l'IA venne dal campo biologico nel 1969: alcuni studenti e ricercatori del Carniege Institute of Technology realizzatono **DENDRAL**[11], un programma capace di ricostruire una molecola semplice a partire dallo spettrometro della massa. Questo nuovo impulso riportò la ricerca nel campo dell'IA in auge fino ad arrivare al 1982 in cui fu creato il primo prodotto commerciale utilizzante l'IA: **R1**[14], utilizzato dalla Digital Equipment, il quale scopo era quello di aiutare a configurare gli ordini per nuovi computer. Dal 1986 in poi l'approccio cambiò strada: cominciarono ad essere utilizzare le reti neurali con **back-propagation**[18] che rivoluzionarono l'approccio e l'apprendimento delle reti rendendolo molto più simile ad una simulazione di ragionamento umano.

1.1.3 IA Senziente

Una volta scoperto l'utilizzo delle reti neurali le applicazioni dell'IA sono state svariate e di diversa natura, poiché si è riusciti a far risolvere i problemi in maniera differente ad una macchina, il che vuol dire che la macchina è capace di prendere delle decisioni diverse in base al problema posto. Per riuscire ad arrivare a conclusioni diverse l'IA deve possedere una conoscenza che può essere pre-caricata oppure imparata con l'esperienza. Uno dei primi esempi di conoscenza acquisita tramite esperienza fu il simulatore di giocatore di scacchi **Deep Blue**[2] che dopo un po' di partite è riuscito a vincere una partita contro il campione di scacchi dell'epoca Garry Kasparov.
Per riuscire a creare algoritmi che imparassero ciò che servisse è nato un settore specifico chiamato "rappresentazione della conoscenza"[26] che si occupa di studiare le possibilità di ragionamento dell'essere umano e come queste possano essere

create in maniera algoritmica e poi essere date ad una macchina, poi soppiantato dal **Machine Learning**, tramite il quale la macchina riesce automaticamente ad imparare dall'esperienza passata.
Una volta raggiunto l'obiettivo di imparare tramite esperienza il passo successivo è creare un'IA simile all'umano nella parte espressiva e motoria. Il primo passo verso ciò furono i **bot** di messaggistica: delle intelligenze artificiali in grado di scrivere e comunicare con un altro essere umano. Inizialmente i bot erano programmati in modo da captare delle parole chiave all'interno delle risposte dell'umano e rispondere con alcune frasi preimpostate; successivamente, aumentando la complessità degli algoritmi, sono stati creati dei metabot che imparassero a comunicare studiando delle conversazioni umane.
Il passo successivo fu la creazione di **EA** (Embodied Agent),[19] un agente intelligente con un corpo fisico in grado di poter interagire con l'ambiente circostante. Il corpo fisico può essere simile all'essere umano o ad un animale, in base alle necessità in quel momento. La maggiore applicazione di questi agenti comunque rimane virtuale, nonostante il nome possa trarre in inganno.
L'evoluzione degli EA sono gli **ECA** (Embodied Conversational Agent)[3] che sono forme intelligenti di interfaccia utente in grado di comunicare fra loro o con esseri viventi tramite linguaggio verbale e non verbale, unendo quindi la parola ad espressioni facciali, gesti e parole in modo che la comunicazione uomo-macchina risulti più naturale e più ricca. Lo stato dell'arte è al **SECA** (Sentient Embodied Conversational Agent)[24][23] che è definito come un ECA in grado di coinvolgere l'utente in conversazioni strutturate e avere alcune qualità senzienti simili a quelle umane in modo che possa percepire e 'sentire' determinate emozioni e rispondere ad esse. I SECA sono composti da più modelli:

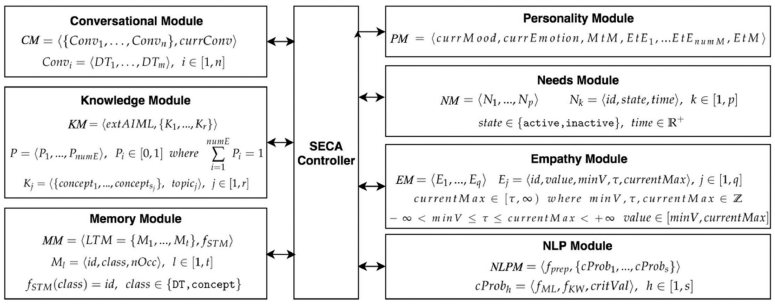


Figura 1.2: Architettura libreria SECA

- **Modulo conversazionale**: consente ai SECA di impegnarsi in conversazioni complesse con gli utenti supportando diverse conversazioni strutturate in linguaggio naturale, definito con una FSM gerarchica con diversi Dialogue Type usati per conversare;

- **Modulo di conoscenza**: gestisce la conoscenza statica associata alle conversazioni mirate, formalizzato con tuple di extAIML e un set di "componenti di conoscenza" già definito;
- **Modulo di personalità**: viene definita una personalità fissa e poi diversi mood ed emozioni che possono variare, lo switch è proporzionale (se si è in "happy" ma succede qualcosa da "crying" si passa a "sad"), questo viene gestito con diverse matrici;
- **Modulo di memoria**: ha il compito di memorizzare informazioni dinamiche per evitare ripetitività e infondere naturalezza alle conversazioni, è divisa in memoria a breve e lungo termine;
- **Modulo dei bisogni**: gestisce e mette in luce le diverse esigenze che possono avere gli agenti;
- **Modulo dell'empatia**: indicatori con valori tra un minimo ed un massimo, può essere quasi gestito tutto tramite questo modulo.
- **Modulo dell'empatia**: indicatori con valori fra un min ed un max, può essere quasi gestito tutto tramite questo modulo.

1.1.4 Applicazioni

Le applicazioni che sfruttano l'Intelligenza Artificiale sono molteplici e di diversa natura. Alcune sono state citate direttamente o indirettamente nei paragrafi precedenti.
L' **Intelligent Data Processing** è un sistema comprendente algoritmi che analizzando dati specifici è in grado di riconoscere, classificare ed estrarre informazioni. Le varie funzionalità saranno implementate in base alla composizione dei dati, prendendo come esempio che i dati siano dei documenti testuali una possibile pipeline di lavoro dell'intelligenza artificiale potrebbe essere composta nel seguente schema: il primo passaggio del processo consisterebbe nel classificare il tipo di documento in fase di elaborazione e bisognerebbe determinare l'inizio e la fine del documento; la tipologia di documento potrebbe essere digitale o elettronica, questa classificazione potrebbe essere eseguita utilizzando la tecnologia OCR (Optical Character Recognition) [15] che è costruita utilizzando algoritmi di machine learning. Il software OCR riconosce caratteri e simboli su un documento, scansiona immagini e fotografie ed è addestrato in quasi 190 lingue per interpretare i dati scansionati dal documento. Altre tecnologie di riconoscimento potrebbero essere Barcode Recognition, ICR (Intelligent Character Recognition) e OMR (Optical Mark Recognition). Una volta classificato il documento, il passaggio successivo e più significativo del processo consisterebbe nell'estrarre informazioni preziose dal documento. Una volta raccolte le informazioni, queste verrebbero inserite nel database richiesto o archiviate di

conseguenza per un utilizzo futuro. L'estrazione dei dati è alimentata da strumenti di corrispondenza dei modelli come Regex (espressioni regolari). Questi strumenti identificano dati specifici nel documento e li presenterebbero in un formato digitale facilmente accessibile. Applicazioni come un sistema di contabilità, sistema ERP (Enterprise Resource Planning), sistema ECM (Enterprise Content Management) e sistema CRM (Customer Relationship Management) trarrebbero grandi vantaggi dall'utilizzo di tale tecnologia. Il passaggio successivo dell'elaborazione intelligente dei dati consisterebbe nell'esportare automaticamente le informazioni e le immagini nei processi aziendali o nei flussi di lavoro. Queste informazioni sarebbero quindi disponibili all'utente.

I **Chatbot** sono dei software programmati in modo da poter simulare una conversazione con un essere umano attraverso l'uso della voce o di testo. L'evoluzione sta andando verso la creazione di vere e proprio interfacce utente, chiamate interfacce conversazionali, capaci di fornire dei servizi e di interfacciarsi con l'utente attraverso domande e risposte molto articolate all'interno di una conversazione o di una chat. Quello che li rende molto efficienti è la capacità di immagazzinare dati ed analizzarli in realtime in modo da poter profilare l'utente e fornire l'aiuto necessario.

In circolazione ora ci sono diverse tipologie di chatbot; negli ambienti del marketing o di servizio al cliente sono molto diffusi dato che riescono ad espletare alcune funzioni in maniera molto efficiente come: offrire prodotti o scontistica ad un determinato cliente analizzando la sua profilazione; rispondere in real time a domande e richieste dell'utente in modo da dare informazioni o servizi; fornire all'utente notizie o aggiornamenti in maniera tempestiva in base alle richieste delle stesso; fornire supporto in base al feedback degli utenti.

I **Recommender System**[17] sono dei sistemi in grado di filtrare le informazioni e che cercano di prevedere la valutazione o la preferenza che l'utente darebbe ad un elemento. I recommender system sono utilizzati in una varietà di aree, con esempi comunemente riconosciuti che assumono la forma di generatori di playlist per servizi video e musicali, consiglieri di prodotti per negozi online o consiglieri di contenuti per piattaforme di social media e consiglieri di contenuti web aperti. Questi sistemi possono funzionare utilizzando un singolo input, come la musica, o più input all'interno e attraverso piattaforme come notizie, libri e query di ricerca. Esistono anche recommender system popolari per argomenti specifici come ristoranti e appuntamenti online. Sono stati inoltre sviluppati recommender system per esplorare articoli di ricerca ed esperti, collaboratori, e servizi finanziari.

I recommender system di solito fanno uso di filtraggio collaborativo e/o filtraggio basato sui contenuti (noto anche come approccio basato sulla personalità), così come altri sistemi come i sistemi basati sulla conoscenza. Gli approcci di filtraggio collaborativo costruiscono un modello dal comportamento passato di un utente (articoli acquistati o selezionati in precedenza e / o valutazioni numeriche date a quegli articoli) così come decisioni simili prese da altri utenti. Questo modello viene quindi

utilizzato per prevedere gli elementi (o le valutazioni degli articoli) a cui l'utente potrebbe essere interessato. Gli approcci di filtraggio basati sul contenuto utilizzano una serie di caratteristiche distinte e pre-etichettate di un elemento per consigliare elementi aggiuntivi con proprietà simili. Gli attuali sistemi di raccomandazione combinano in genere uno o più approcci in un sistema ibrido. I recommender system sono un'alternativa utile agli algoritmi di ricerca poiché aiutano gli utenti a scoprire elementi che altrimenti non avrebbero trovato. Da notare, i recommender system sono spesso implementati utilizzando motori di ricerca che indicizzano dati non tradizionali.
Il **Natural Language Processing**[12] è il ramo dell'informatica che si occupa di dare ai calcolatori la capacità di comprendere il testo e le parole pronunciate in maniera simile agli esseri umani. Il natural language processing combina la linguistica computazionale - modellazione basata su regole del linguaggio umano - con modelli statistici, di apprendimento automatico e di apprendimento profondo. Insieme, queste tecnologie consentono ai computer di elaborare il linguaggio umano sotto forma di testo o dati vocali e di "comprenderne" il pieno significato, completo delle intenzioni e dei sentimenti di chi parla o scrive. Il natural language processing gestisce programmi per computer che traducono il testo da una lingua all'altra, rispondono ai comandi vocali e riassumono rapidamente grandi volumi di testo, anche in tempo reale. Il linguaggio umano è pieno di ambiguità che rendono incredibilmente difficile scrivere software che determini accuratamente il significato previsto del testo o dei dati vocali. Omonimi, omofoni, sarcasmo, modi di dire, metafore, grammatica e eccezioni d'uso, variazioni nella struttura della frase: queste solo alcune delle irregolarità del linguaggio umano che impiegano anni per imparare, ma che i programmatori devono insegnare alle applicazioni basate sul linguaggio naturale a riconoscere e capire accuratamente fin dall'inizio se queste applicazioni saranno utili. Diverse attività di natural language processing scompongono il testo umano ed i dati vocali in modi che aiutano il computer a dare un senso a ciò che sta ingerendo. Queste attività includono: il riconoscimento vocale, l'etichettatura vocale, la disambiguazione del senso delle parole, la risoluzione del riferimento e l'analisi del sentimento. Il natural language processing viene utilizzato anche in alcuni chatbot.
La **Computer Vision** è un campo interdisciplinare che si occupa di come i computer possono essere realizzati per acquisire una comprensione di alto livello da immagini o video digitali. Dal punto di vista dell'ingegneria, cerca di automatizzare i compiti che il sistema visivo umano può svolgere.Implica lo sviluppo di una base teorica e algoritmica per ottenere la comprensione visiva automatica. La computer vision si occupa della teoria alla base dei sistemi artificiali che estraggono informazioni dalle immagini. I dati dell'immagine possono assumere molte forme, come sequenze video, visualizzazioni da più telecamere o dati multidimensionali da uno scanner medico. Come disciplina tecnologica, la computer vision cerca di applicare le sue teorie e modelli per la costruzione di sistemi di visione artificiale.

Le attività di visione artificiale includono metodi per acquisire, elaborare, analizzare e comprendere immagini digitali e l'estrazione di dati ad alta dimensione dal mondo reale al fine di produrre informazioni numeriche o simboliche, ad esempio sotto forma di decisioni. La comprensione, in questo contesto, significa la trasformazione delle immagini visive (l'input della retina) in descrizioni del mondo che possono interfacciarsi con altri processi di pensiero e sollecitare un'azione appropriata. Questa comprensione dell'immagine può essere vista come il districare le informazioni simboliche dai dati dell'immagine utilizzando modelli costruiti con l'ausilio di geometria, fisica, statistica e teoria dell'apprendimento. Seguono alcuni problemi classici che possono essere risolti tramite computer vision:

- riconoscimento: determinare se un oggetto, un'azione o un'attività siano all'interno di un'immagine o di un video;[20][25]
- analisi del moto: elaborazione di una sequenza di immagini per produrre una stima della velocità in ogni punto dell'immagine o nella scena 3D, o anche della telecamera che produce le immagini;[9]
- ricostruzione di scene: data una o più immagini di una scena, o di un video, la ricostruzione della scena mira a calcolare un modello 3D della scena; nel caso più semplice il modello può essere un insieme di punti 3D mentre con metodi più sofisticati producono un modello di superficie 3D completo; [27]
- ricostruzione di immagini: è l'operazione di prendere un'immagine corrotta/-rumorosa e stimare l'immagine pulita e originale; la corruzione può presentarsi in molte forme come sfocatura di movimento, rumore e messa a fuoco errata della fotocamera.

L' **Intelligent Object** è un oggetto che migliora l'interazione non solo con le persone ma anche con altri intelligent object. Sono prodotti, risorse e altre cose incorporate con processori, sensori, software e connettività che consentono lo scambio di dati tra il prodotto e il suo ambiente, produttore, operatore/utente, e altri prodotti e sistemi. La connettività consente inoltre ad alcune funzionalità del prodotto di esistere al di fuori del dispositivo fisico, in quello che è noto come il cloud del prodotto. I dati raccolti da questi prodotti possono quindi essere analizzati per informare il processo decisionale, consentire efficienze operative e migliorare continuamente le prestazioni del prodotto. Ci si riferisce non solo all'interazione con il mondo fisico ma anche con il mondo virtuale (ambiente informatico). Un intelligent virtual object può essere creato come artefatto o prodotto fabbricato o incorporando tag elettronici come tag o sensori RFID in oggetti fisici non intelligenti. Gli intelligent virtual object vengono creati come oggetti software intrinseci durante la creazione e il funzionamento di una simulazione o di un gioco del mondo virtuale o cibernetico. Il concetto di intelligent object ha diverse origini e usi: ogni applicazione di intelligent object ha carattesristiche diverse in base all'ambiente ed

all'oggetto che si stanno prendendo in considerazione.
Un **Autonomous Robot** è un robot che esegue comportamenti o attività con un alto grado di autonomia (senza influenze esterne). Il primo requisito per una completa autonomia fisica è la capacità di un robot di prendersi cura di se stesso. Molti dei robot alimentati a batteria oggi sul mercato possono trovare e connettersi a una stazione di ricarica e alcuni giocattoli come Aibo di Sony sono in grado di agganciarsi automaticamente per caricare le batterie. L'auto-manutenzione si basa sulla "propriocezione", ovvero la percezione del proprio stato interno. Nell'esempio di ricarica della batteria, il robot può dire in modo propriocettivo che le sue batterie sono scariche e quindi cerca il caricabatterie. Un altro sensore propriocettivo comune è per il monitoraggio del calore. Sarà necessaria una maggiore propriocezione affinché i robot lavorino in modo autonomo vicino alle persone e in ambienti difficili. I sensori propriocettivi comuni includono il rilevamento termico, ottico e tattile, nonché l'effetto Hall (elettrico). Il secondo requisito è quello di percepire l'ambiente esterno (esterocezione). I robot autonomi devono disporre di una gamma di sensori ambientali per svolgere il loro compito. I sensori esterocettivi comuni includono lo spettro elettromagnetico, il suono, il tatto, la sostanza chimica (odore, odore), la temperatura, la portata di vari oggetti e l'altitudine. Il passo successivo nel comportamento autonomo è eseguire effettivamente un compito fisico. Una nuova area che mostra una promessa commerciale è quella dei robot domestici, con una marea di piccoli robot aspirapolvere che iniziano con iRobot ed Electrolux nel 2002. Sebbene il livello di intelligenza non sia elevato in questi sistemi, navigano su vaste aree e pilotano in situazioni ristrette intorno alle case utilizzando sensori di contatto e senza contatto. Entrambi questi robot utilizzano algoritmi proprietari per aumentare la copertura rispetto al semplice rimbalzo casuale. Il livello successivo di prestazioni di attività autonome richiede che un robot esegua attività condizionali.

1.2 Animazione

1.2.1 Introduzione

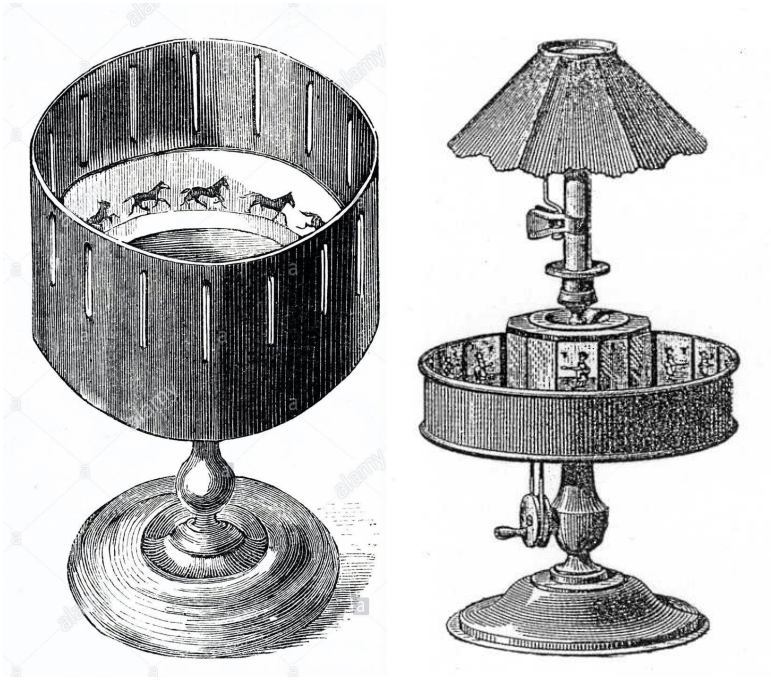
Con animazione si intende una tecnica in grado di far percepire all'utente il movimento tramite delle immagini visualizzate in rapida successione. Per far si che si abbia una percezione dell'immagine in movimento di deve superare il tempo di percezione dell'occhio umano, cioè circa 10 fps (fotogrammi al secondo).
La percezione del movimento nell'animazione viene data dal concetto di **movimento apparente a corto raggio**: partendo dal concetto di movimento apparente possiamo definire la percezione di un movimento nonostante il movimento vero e proprio non esista, questo avviene tramite più stimoli intervallati temporalmente, questo intervallo sarà scelto in funzione della distanza fra gli stimoli; le componenti più importanti di questa percezione sono l'asincronia di inizio stimolo (la tempistica tra l'inizio di uno stimolo e il successivo) e l'intervallo interstimolare (fine di uno e inizio dell'altro); con la definizione di "corto raggio" invece si intende il fatto che le tempistiche saranno molto basse e lo spostamento fra uno stimolo e l'altro sarà molto ridotto.
Un esempio per il movimento a corto raggio è il cinema che ha un intervallo fisso di sequenza di stimoli (24 fps) e uno spostamento minimo degli oggetti in ciascun fotogramma tra 5 e 15 arco/minuti.

1.2.2 Storia

L'origine dell'animazione non può essere definita con certezza dato che già all'interno delle cave paleolitiche sono state trovati dei disegni raffiguranti animali con più zampe in posizioni sovrapposte o in serie che possono essere interpretate come lo stesso animale in posizioni diverse. Queste pitture primordiali venivano animate tramite una luce del fuoco tremolante o con una torcia, animando solo una determinata parte della roccia il disegno veniva percepito in movimento.
Altre tipologie di animazione preistorica sono state ritrovate in vari reperti archeologici: dischi con un foro al centro e disegni da entrambi i lati, ciotole con sequenze di immagini all'interno e murali egiziani che rappresentano serie lunghe di immagini sequenziali.
I cinesi, sin da prima del 1000 DC, avevano creato una lanterna rotante che aveva sagome proiettate sui suoi lati di carta sottili che sembravano inseguirsi a vicenda. Questa era chiamata "lampada del cavallo al trotto" in quanto rappresenterebbe tipicamente cavalli e cavalieri. Le sagome ritagliate erano attaccate all'interno della lanterna a un albero con una girante a palette di carta in cima, ruotata dall'aria riscaldata che saliva da una lampada.
La data principale che ha portato ad una creazione di strumenti di animazione è il

1671 nel quale Athanasius Kircher scrisse della **Lanterna Magica**, il primo proiettore di immagini fisse.
Da inizio a fine 1800 furono creati diversi strumenti in grado di creare la percezione di animazione di sequenze di immagini:

- **taumatropio** (1824): consiste in un dischetto con un'immagine per lato che viene fatto ruotare da due fili appesi all'estremità e crea illusione di movimento;
- **fenachistoscopio** (1832): dispositivo costituito da due dischi, uno dei quali con finestre equidistanti dalle quali l'utente può guardare il secondo disco che, invece, contiene una sequenza di immagini;
- **zootropio** (1834): una striscia di carta con una serie di disegni è posta all'interno di un cilindro dotato di feritoie a intervalli regolari, ognuna di queste feritoie viene utilizzata per visualizzare una sola immagine;
- **cineografo** (1868): un libro con pagine disegnate, sfogliandolo rapidamente si può visualizzare una breve animazione;
- **prassinoscopio** (1877): evoluzione del zootropio poiché invece delle feritoie costituito da specchi a 45° in modo da riflettere le immagini verso l'osservatore.



(a) Zootropio (b) Prassinoscopio

Figura 1.3: Strumenti di fino '800 per animazioni

Il primo animatore della storia è considerato Charles-Émile Reynaud con la creazione del **teatro ottico** nel 1888: il sistema consiste in una serie di lastre di vetro dipinte a mano e montate su bande di pelle; le bande erano collegate fra loro attraverso dei nastri metallici forati che si agganciavano all'ingranaggio del tamburo ruotante, in modo da venire allineati alla lanterna del proiettore.
La creazione più importante fu quella del **cinematografo** dei fratelli Lumière nel 1894, da quel momento in poi vennero costruite animazioni come le conosciamo attualmente. Una volta definita la tecnica di fotogramma per fotogramma furono inventate diverse tecniche per la creazione di animazioni diverse fra loro:

- **stop motion**: la tecnica è attuata con oggetti reali che vengono fotografati in posizioni diverse, mettendo fotogramma per fotogramma le foto viene creata un'animazione;
- **animazione di silhouette**: tecnica che si basa sulla creazione di figure nere riprese di profilo poi poste su un piano orizzontale e retroilluminato in modo da conferire opacità al nero delle figure e luminosità agli sfondi colorati;
- **fogli di celluloide**: sottili fogli di celluloide all'interno dei quali vengono disegnati i fondali ed i personaggi animati; se fatti scorrere difronte ad una macchina da presa danno l'illusione di movimento.

Lo step successivo è definito dalla creazione dell'animazione digitale.

1.2.3 Animazione digitale

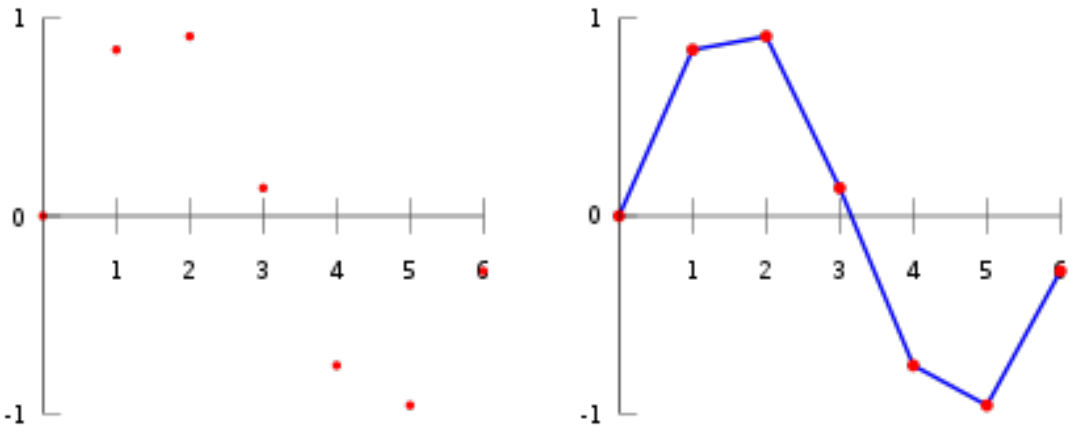
L'animazione ditale è il processo utilizzato per creare digitalmente delle immagini animate. La moderna animazione digitale utilizza principalmente la computer grafica 3D per creare un'immagine bidimensionale, anche se ancora viene utilizzata anche la grafica 2D che permette tempi di render molto più brevi.
Il campo delle animazioni digitali è in costante cambiamento e innovazione: il problema principale che viene affrontato è la creazione di un workflow che porti al risultato finale in maniera più veloce e, in alcuni casi, più realistica. Il tipo di workflow dipende dal tipo di applicazione che riguarda l'animazione e le possibili applicazioni sono svariate, passano da film d'animazione ad esperienze di realtà virtuale, e la differenza fondamentale fra loro è il dispositivo tramite il quale si fruirà l'animazione ed il fatto che essa sia prerenderizzata oppure in realtime.
Nella seguente sezione presenteremo quelle che sono le principali soluzioni utilizzate in questo abito:

- **keyframing**,
- **motion capture**,
- **machine learning**.

Keyframing

Il keyframing è la tecnica base per le animazioni tradizionali: si basa sulla definizione di alcune pose chiave in specifici frame dell'animazione (keyframe) che definiscono i punti di inizio e di fine di qualsiasi transizione graduale. L'animazione finale viene ottenuta tramite **interpolazione** (tweening) dei vari keyframe. Spesso è necessario un ulteriore lavoro di perfezionamento degli in-between frames per ottenere delle animazioni il più fluide possibili o perché il lavoro di interpolazione risultante non è soddisfacente e si voglia personalizzare l'animazione.
Esistono numerose tecniche di interpolazione, a seconda delle necessità:

- **lineare**: data una sequenza di n numeri reali *distinti* x_k per $k = 1,..,n$ chiamati nodi e per ogni x_k sia dato un secondo numero reale y_k tale per cui: $f(x_k) = y_k$ per $k = 1,...,n$.



(a) Punti da interpolare (b) Interpolazione lineare

Figura 1.4: Step per l'interpolazione lineare

- **polinomiale**: dati n *+1* punti (x_i,y_i) distinti dove $y = f(x)$, bisogna determinare il polinomio di grado minimo che passi per i punti assegnati.

$$y_k = f(x_k) \text{ con } (x_i,y_i),\ i = 0,1,...,n$$

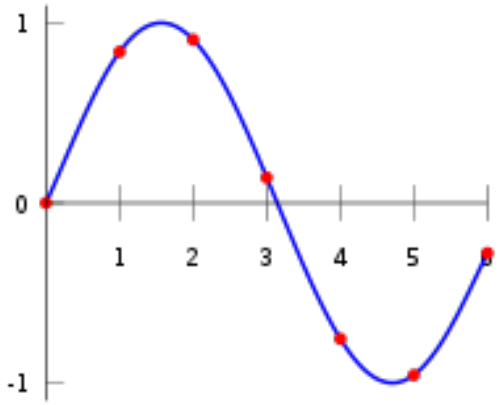


Figura 1.5: Interpolazione polinomiale

Esistono specializzazioni di interpolazione polinomiale come: funzione spline, spline cubica di Hermite e curva di Bezier.
Nel caso di animazioni riguardanti un modello umanoide è necessario modificare la posizione di ogni osso dell'armatura e modificare i valori di eventuali blend shapes.

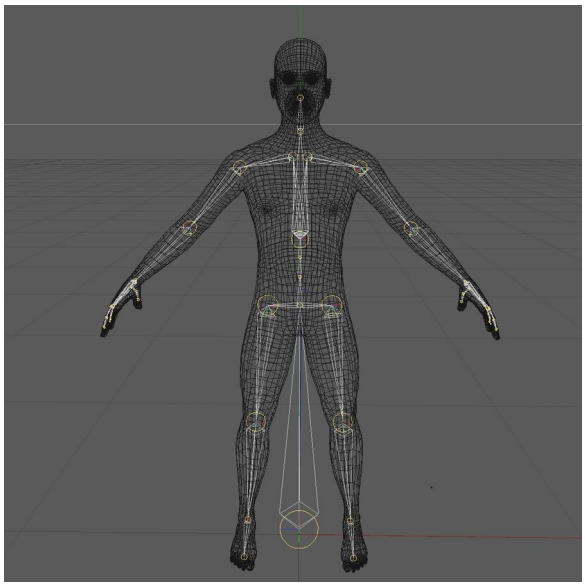


Figura 1.6: Rig di un modello umanoide

Il problema principale di questo tipo di workflow è che ogni animazione deve essere fatta su misura per ogni tipo di interazione con l'ambiente in cui si trova il personaggio ed è raro che un'animazione possa essere riutilizzata senza modifiche

anche prendendone una diversa in minima parte. Questo comporta la necessità di intervenire manualmente ogni qualvolta debba essere creata un'animazione.

Motion capture

Con motion capture (mocap) si intende il processo di registrazione del movimento di oggetti o persone. Nell'ambito dell'animazione con motion capture ci si riferisce alla registrazione di attori umani o, in più rari casi, di animali, le informazioni che vengono catturare servono ad animare dei personaggi digitali.
Inizialmente l'utilizzo del mocap fungeva da analisi fotogrammetrica nella ricerca biomeccanica (anni '70 e '80) per poi espandersi negli altri ambiti.
I **sistemi ottici** utilizzano i dati acquisiti dai sensori di immagine per triangolare la posizione 3D di un soggetto tra due o più telecamere calibrate per fornire proiezioni sovrapposte. L'acquisizione dei dati può essere attuata grazie ad un artista indossante dei marker (che possono essere di tipi diversi) vicino a ciascuna articolazione in modo che si identifichi il movimento tramite le posizioni e gli angoli tra i marker. Questi sistemi producono dati con tre gradi di libertà per ogni marker e le informazioni rotazionali devono essere dedotte dall'orientamento relativo di tre o più marker; per esempio indicatori di spalla, gomito e polso che forniscono l'angolo del gomito. Per fare in modo che i marker vengano tracciati in maniera ottimale bisogna far sì che la velocità di tracciamento sia almeno due volte il tasso di frequenza del movimento desiderato.
Esistono anche vari sistemi che riescono ad ottenere le informazioni di mocap anche senza marker ma semplicemente estraendo la sagoma del perfomer dallo sfondo e poi calcolare le articolazioni tramite un modello matematico.
Nel caso in cui si abbiano più artisti e l'area sia molto grande l'acquisizione viene eseguita con l'aggiunta di più camere ed è conveniente utilizzare dei sistemi ibridi in grado di combinare i sensori inerziali con i sensori ottici in modo da ridurre l'occlusione (molto problematica nel caso di più artisti), aumentare il numero di utenti e migliorare la capacità di tracciamento.
I marker che vengono utilizzati nei sistemi ottici possono essere principalmente di due tipologie:

- **passivi**: utilizzano marker rivestiti con un materiale in grado di riflettere la luce generata vicino alla telecamera; la soglia della telecamera deve essere regolata in modo che vengano campionati solo i marker riflettenti luminosi, ignorando possibili altri materiali (pelle o tessuti);

- **attivi**: anziché riflettere la luce generata esternamente, i marker stessi sono alimentati per emettere la propria luce; triangolano le posizioni illuminando un LED alla volta molto rapidamente o più LED e vengono identificati in base alla loro posizione relativa.

Esistono anche altre varianti di marker ma che hanno alla base una di queste due tipologie.
Grazie alla computer vision si riesce ad avere dei sistemi ottici senza marker in grado di ottenere informazioni di mocap.

Figura 1.7: Mocap con sistema ottico

Gli approcci con sistemi non ottici possono essere:

- **sistemi inerziali**: tecnologia basata su sensori inerziali miniaturizzati, modelli biomeccanici e algoritmi di fusione dei sensori; i dati che vengono catturati dai sensori sono trasmessi ad un computer che tramite un software registra e visualizza il movimento;
- **movimento meccanico**: tramite una struttura direttamente collegata al corpo vengono tracciati gli angoli delle articolazioni;
- **sistemi magnetici**: calcolano la posizione e l'orientamento in base al flusso magnetico relativo di tre bobine ortogonali; sono dei sistemi a sei gradi di libertà;
- **sensori di allungamento**: sono condensatori a piastre parallele flessibili che misurano l'allungamento, la flession il taglio o la pressione.

Machine learning

All'interno si mondi virtuali diventa difficile e dispendioso fare in modo che i personaggi possano interagire in maniera realistica con ogni oggetto presente poiché le animazioni, per essere precise, dovrebbero essere modulate per ogni oggetto presente in maniera manuale. L'utilizzo di machine learning è in grado di semplificare e automatizzare il processo di adattamento agli oggetti circostanti. Tramite un allenamento di rete neurale un personaggio può riuscire a compiere dei task all'interno di un mondo virtuale, facendo in modo che il personaggio conosca l'ambiente che

lo circonda e gli oggetti che lo compongono, si possono creare delle animazioni che riescano ad essere molto precise.
Un esempio di creazione di animazioni tramite machine learning è lo studio presentato durante il SIGGRAPH ASIA 2019 all'interno del quale le animazioni sono gestite tramite una **Neural State Machine**[22] costituita da una *rete di previsione del moto* e da una *rete di gating*. La rete di previsione del moto è il componente principale responsabile dell'auto-regressione delle informazioni relative al personaggio mentre la rete di gating ha il compito di decidere i coefficienti di miscelazione dei pesi prendendo in considerazione l'obiettivo che il personaggio deve raggiungere e le varie fasi del movimento per generare in maniera dinamica la rete di previsione del moto.
La rete di previsione del moto è composta principalmente da due moduli: il modulo di codifica che riceve i componenti dello stato del personaggio dei frame precedenti e li codifica individualmente utilizzando una rete di tre layer e il modulo di predizione che, invece, riceve gli output del primo modulo e predice lo stato del personaggio nel frame corrente.
La rete di gating è una rete neurale fully-connected i quali output rappresentano i coefficienti di blending.

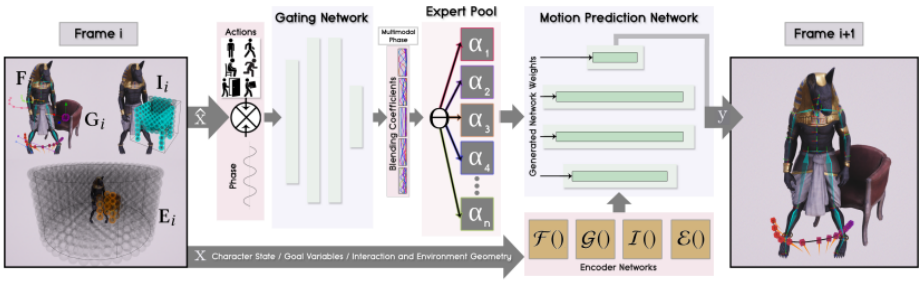


Figura 1.8: Schema della Neural State Machine

L'utilizzo del machine learning è molto funzionale nel caso in cui le animazioni del personaggio preso in considerazione siano molto realistiche e si abbiano dei computer molto performanti visto che i calcoli da fare sono abbastanza lunghi e dispendiosi. Nel caso in cui si voglia creare un proprio stile nella creazione dell'animazione questa tipologia di approccio risulterebbe più complessa da gestire.

1.3 Sintesi Vocale

1.3.1 Introduzione

La sintesi vocale è la produzione artificiale della voce umana. Questo tipo di produzione può essere effettuato sia via software che via hardware, dipendentemente dai risultati che si vogliano ottenere e dagli strumenti a disposizione.
La sintesi crea un **segnale audio**, che è una rappresentazione analogica di un suono con diverse caratteristiche tipiche di ogni tipo di segnale.

- *larghezza di banda*: la misura dell'ampiezza di banda dello spettro;
- *livello nominale*: il livello operativo al quale un dispositivo di elaborazione del segnale elettronico è progettato per funzionare; i circuiti elettronici sono limitati dal segnale massimo che possono gestire e nel rumore elettronico generato internamente che va aggiunto al segnale;
- *livello di potenza*: viene descritto tramite l'unità di misura decibel (dB) che viene utilizzata per esprimere il rapporto tra un valore di potenza o di radice rispetto ad un altro valore, su una scala logaritmica.

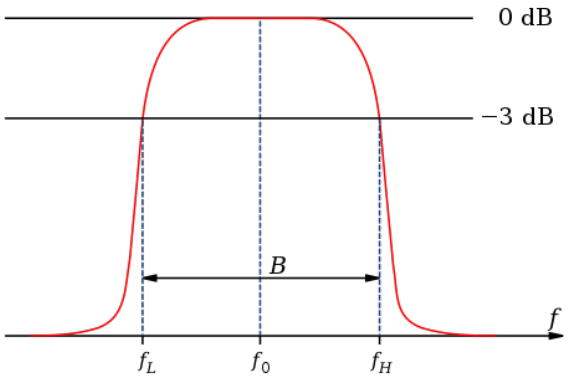


Figura 1.9: Banda dello spettro (B)

- *tensione*: indica la differenza tra il potenziale elettrico in due punti dello spazio.

Le tipologie di creazione sono svariate ma tutte partono da una base definita: un database contenente parole o frasi di riferimento. Una volta creato o ottenuto il database si possono concatenare direttamente alcune parole oppure fare in modo che venga creata ex novo una voce analizzando delle caratteristiche della voce umana tali da poter creare una voce totalmente sintetica.

Un sistema si sintesi vocale è composto da due elementi principali: la parte front-end esegue la parte di pre-elaborazione della voce, assegna quindi le trascrizioni fonetiche a ciascuna parola e divide e contrassegna il testo in unità prosodiche, come frasi e proposizioni; come output della parte front-end abbiamo la trascrizione fonetica e le informazioni di prosodia; il back-end invece funge da sintetizzatore, cioè converte la rappresentazione linguistica (quindi l'outuput del front-end) in suono.
Nella seguente sezione analizzeremo le varie metodologie utilizzate per la sintesi vocale.

1.3.2 Storia

Già molto prima che si scoprisse l'elettricità e l'attuale elaborazione dei segnali l'uomo tentò di costruire dei macchinari che riuscissero a sintetizzare la voce umana, in particolare fra il X e XIII secolo.
Il primo vero e proprio meccanismo in grado di riprodurre dei fonemi vocali fu inventato nel 1779 da Christian Kratzenstein; seguirono diverse macchine che ampliarono quella di Kratzenstein fino ad arrivare al brevetto del **Vocoder** nel 1935 depositato da Homer Dudley [4]. Il Vocoder è in grado di elaborare dei segnali audio tramite degli algoritmi di codifica. Questa creazione fu un passo avanti nella codifica del segnale audio e il risultato di output creò delle parole intellegibili. Il successivo passo fu la creazione di uno strumento, chiamato **Pattern playback**, capace di creare uno spettrogramma dato un segnale audio, tramite questo si scoprirono segnali acustici per la percezione di segmenti fonetici.
I primi sistemi di sintesi vocale basati su computer nacquero alla fine degli anni '50, ma i più importanti furono due: nel 1961 John Larry Kelly, Jr e il suo collega Louis Gerstman usarono un computer IBM 704 per sintetizzare il parlato e nel 1968 Noriko Umeda sviluppò il primo sistema di sintesi vocale in inglese generale nel 1968.
Una delle scoperte fondamentali per la creazione di sistemi di text-to-speech fu la **codifica lineare predittiva** nel 1966, una forma di codifica vocale, che fu la base per la creazione dei primi chip per sintetizzatori vocali.
Il **metodo delle coppie spettrali di linea** [7] è stato sviluppato nel 1975 e viene utilizzato per la codifica del parlato ad alta compressione. Questo metodo è considerato un'importante tecnologia per la sintesi vocale e la codifica considerando che negli anni '90 è stato adottato da quasi tutti gli standard internazionali di codifica vocale come componente essenziale, contribuendo al miglioramento della comunicazione vocale digitale su canali mobili e Internet.
La tecnologia **DECTalk** creata nel 1984 fu un sistema dominante per tutti gli anni '80 e '90 fino ad arrivare alla capacità di creare linguaggio multilingua facendo ampio uso di metodi di elaborazione del linguaggio naturale.

Tecnologie di sintetizzazione

- La *sintesi concatenativa* è una tecnica che sfrutta la concatenazione di brevi campioni di suono registrato per sintetizzare nuovi suoni Questo tipo di tecnologia può essere definita guidata, poiché si basa su suoni definiti a priori.
- La *sintesi delle formanti* è un tipo di sintetizzazione che utilizza la sintesi additiva e un modello acustico. I parametri principali come la frequenza fondamentale, la voce ed i livelli di rumore vengono variati nel tempo in modo da creare una forma d'onda di discorso artificiale. Questo tipo di tecnologia sintetizza delle frasi non simili alla voce umana (risultando molto robotica).
- La *sintesi articolatoria* fa riferimento al tratto vocale umano ed ai processi di articolazione che vi si verificano per creare dei modelli. In pratica il parlato viene creato simulando digitalmente il flusso d'aria che attraversa il tratto vocale umano.
- L' *HMM-based synthesis* è basata sul modello di Markov nascosto; in questo sistema lo spettro di frequenza, la frequenza fondamentale e la durata del discorso sono modellati simultaneamente dagli HMM.

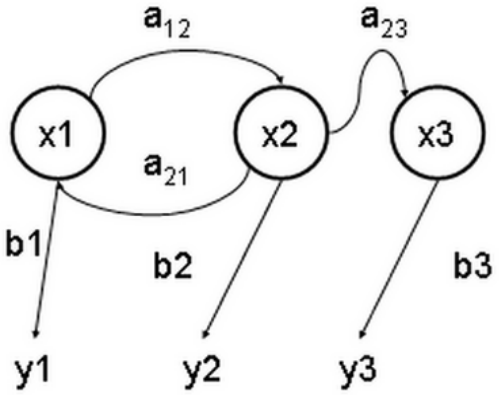


Figura 1.10: Transizioni di stato in un HMM

- La *sintesi dell'onda sinusoidale* è una tecnica per sintetizzare il discorso sostituendo le formanti con fischi di tono puro.
- La sintesi basata su *deep learning* è un tipo di sintetizzazione basata sulle reti neurali.

1.3.3 Deep Learning synthesis

Rete neurale

Con rete neurale artificiale si intende un modello computazionale basato su una serie di algoritmi che tentano di mimare il comportamento delle reti neurali biologiche. Il cervello umano è composto da neuroni che comunicano fra loro e inviano e ricevono informazioni. Queste informazioni vengono ricevute attraverso il dendrite (che può essere considerato come un cavo di ingresso elettronico), elaborate all'interno del nucleo (che può essere considerato come un'unità computazionale elettronica) e poi successivamente inviate ad altri neuroni attraverso l'assone (cavo di uscita). Le reti neurali artificiali utilizzano lo stesso meccanismo del neurone: sono composte da diversi neuroni che ricevono segnali di input che vengono elaborati e poi inviati ai neuroni successivi.

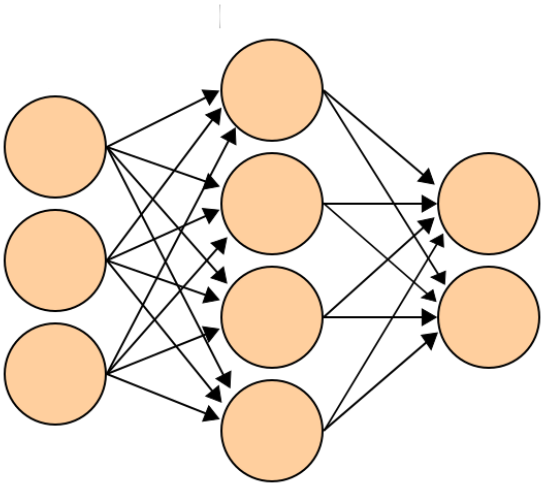


Figura 1.11: Rete neurale artificiale

Una rete neurale, dal punto di vista matematico, è composta da una funzione *f(x)* e definita come una composizione di altre funzioni *g(x)* che possono essere definite a loro volta tramite altre funzioni. Una struttura di reti descrive questo comportamento, con i nodi che definiscono le funzioni *f(x)* e le frecce in ingresso le funzioni *g(x)* che la compongono. Tramite le frecce si definisce anche la dipendenza di un nodo dall'altro.
Ogni unità di computazione ha una matrice di pesi che serve a definire l'importanza di ogni funzione entrante e da computare; ciò che uscirà in output (e che entrerà come input nel neurone successivo) sarà la composizione della matrice di pesi e

funzioni in entrata.
I vari strati di nodi possono essere definiti come **layer**:

- *Input Layer*: l'insieme dei nodi che ricevono gli input dell'allenamento;
- *Hidden Layer*: la funzione applica pesi agli input e li dirige attraverso una funzione di attivazione come output;
- *Output Layer*: nodi che tirano fuori i valori di predizione della rete.

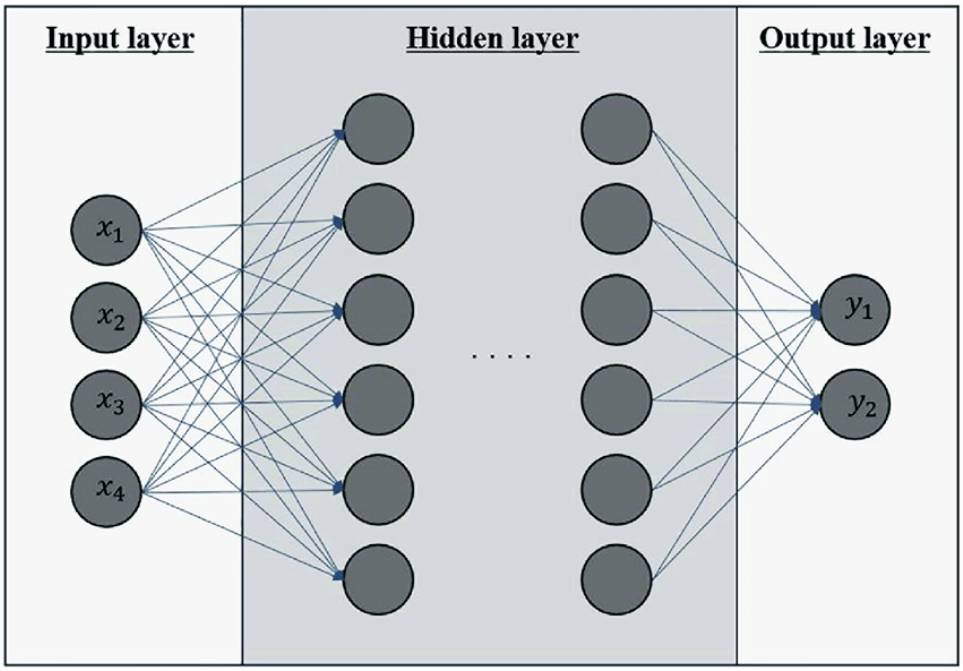


Figura 1.12: Layers in una rete neurale

Una delle più importanti innovazioni nel campo delle reti neurali è sicuramente l'avvento delle **reti convoluzionali** (CNN) che hanno portato a un vero e proprio cambiamento di paradigma e soprattutto alla possibilità di creare delle reti molto più profonde, motivo per cui oggi possiamo parlare di deep learning.

Applicazioni nel campo audio

Il deep learning ha avuto negli ultimi anni un notevole sviluppo soprattutto nel campo visivo e della computer vision, e strumenti di questo tipo sono utilizzati in tantissimi campi, dalla produzione industriale alla sicurezza.

Per poter sfruttare le CNN anche in campo audio è necessario ottenere una rappresentazione visiva del segnale audio. Questa può essere di diverso tipo: come un segnale sinusoidale, o a livello di spettrogramma. Tuttavia, c'è la necessità di distinguere tra parametri oggettivi e soggettivi del segnale, poichè, nella maggior parte dei casi, a una variazione di un qualche parametro oggettivo estratto da un segnale audio non corrisponde una stessa variazione a livello percettivo, e dunque una valutazione su questi parametri non ci fornirebbe una corretta visione delle cose. Per questo motivo si rende necessaria una rappresentazione dell'audio che tenga conto di questi fattori e delle caratteristiche percettive dell'orecchio umano, cioè il destinatario di ciò che viene prodotto dalla rete neurale. A questo scopo viene utilizzato il **Mel-spectrogram**, che non è altro che uno spettrogramma che utilizza la scala di Mel sull'asse y. La scala di Mel infatti è un'unità di misura che ha lo scopo di simulare la percezione non lineare che l'orecchio umano ha rispetto ai suoni, tenendo più in considerazione le basse frequenze e di meno le frequenze più alte, e ci permette dunque di ottenere le informazioni che servono alla rete neurale per produrre dei segnali di buona qualità, almeno a livello percettivo. La scala di Mel ci permette approssima meglio il sistema udito umano rispetto alla classica analisi in frequenza linerare. Gli **MFCC** (Mel Fequency Cepstrum Coefficients) sono infine estratti dal Mel-spectrogram e sono le feature fondamentali che il modello prende in considerazione nel suo allenamento

1.3.4 Esempi di reti neurali per l'audio

- **WaveNet** [16]: rete neurale sviluppata da Google per la generazione di segnali audio. Il modello è basato sulla PixelCNN. In fase di allenamento, ogni campione è condizionato da quello precedente. Riesce a sintetizzare delle voci molto simili a quelle di un essere umano ma un grosso svantaggio è la mancanza di un layer di pooling, quindi la dimensione dell'input deve essere identica a quella dell'output

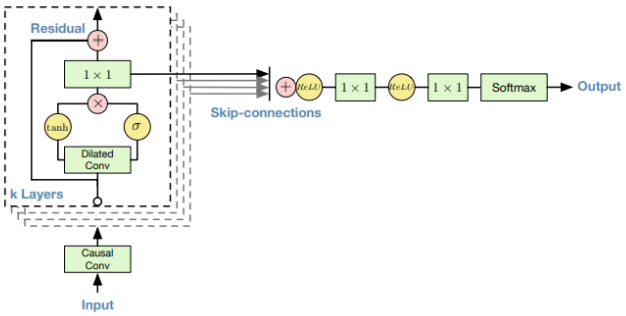


Figura 1.13: Architettura di WaveNet

- **Tacotron [29]:** modello end-to-end sviluppato da Nvidia. Allenato a livello di frame, è più veloce rispetto ai modelli allenati a livello di campioni, come WaveNet. In input viene fornita una coppia di audio e del relativo testo pronunciato e questo rende molto facile estendere i dataset utilizzabili.

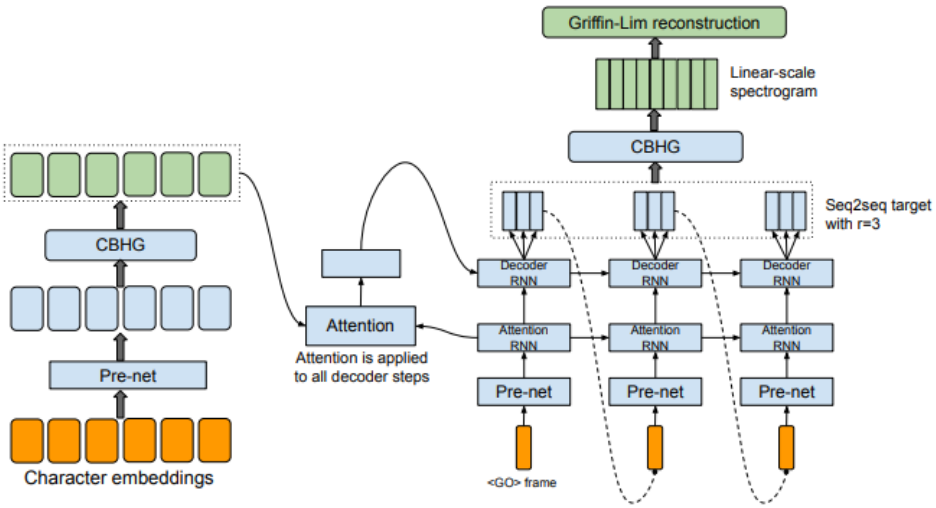


Figura 1.14: Architettura di Tacotron

Capitolo 2

Tools

All'interno di questa sezione saranno descritti i tools esterni che sono stati utilizzati per la creazione delle animazioni e della sintetizzazione vocale.
La scelta dei tools è frutto di una ricerca ben accurata per raggiungere i nostri obiettivi, prendendo in considerazione il compromesso fra prestazioni, facilità di adattamento e velocità di creazione.
Dal punto di vista prestazionale abbiamo considerato il fatto che ci potrebbero essere delle applicazioni costruite con questo framework che inglobano tanti ECA che interagiscono fra loro e con l'ambiente senza che avvenga un calo di frame rate tale da rendere l'applicazione non utilizzabile. Si parlerà più specificatamente dell'argomento all'interno del capitolo dei test.
Riguardo la facilità di adattamento abbiamo considerato la versatilità delle possibili azioni che l'intelligenza artificiale potrebbe effettuare, è un fattore chiave poiché il framework potrebbe essere utilizzato in diversi ambiti e situazioni.
La velocità di creazione, invece, riguarda il fatto che possibili nuovi programmatori si potrebbero approcciare alla modifica e alla creazione di nuove azioni per ogni ECA, abbiamo progettato e utilizzato i tools che possano rendere questa modifica il più veloce e semplice possibile.

2.1 Motion Matching for Unity

Motion Matching for Unity (MxM) è un sistema di animazione alternativo a mecanim che consente un'animazione fluida e reattiva senza la necessità di una macchina a stati. Non è necessario specificare transizioni, definire condizioni o tenere traccia di complesse logiche di animazione.
Motion Matching consente all'animazione di fluire liberamente attraverso l'intero set di animazione, saltando a qualsiasi posa in qualsiasi momento. Le pose dell'animazione vengono scelte sia in base alla posa corrente che all'input di gioco desiderato. Qualunque cosa corrisponda alla posa e all'input desiderato, viene scelto come migliore e viene rapidamente integrato. Bilanciando la posa e l'input desiderato, il motion maching è in grado di ottenere animazioni ad alta fedeltà pur continuando a ottenere una buona risposta di input.
La creazione e scelta della posa si basa sulla posa attuale dell'agente virtuale e la traiettoria futura che percorrerà l'agente. Viene effettuato un calcolo per capire quale sia la posa migliore prendendo in considerazione questi due fattori.
A livello di prestazioni questo tool riesce a mantenere un alto numero di frame rate grazie all'utilizzo dei jobs di Unity e del nuovo burst compiler.

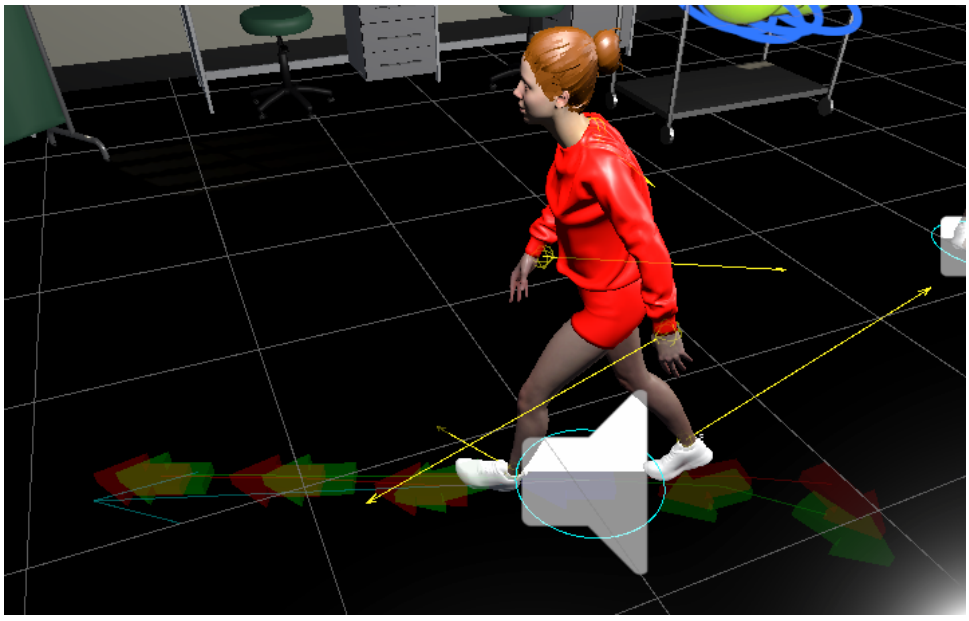


Figura 2.1: Calcolo di traiettoria definito dalle frecce

2.1.1 MxM Pre-Processor

Il componente principale del tool è l'MxM Pre-Processor che contiene tutte le informazioni principali che servono per creare i dati di animazione (Anim Data) che saranno propri di uno specifico agente virtuale.

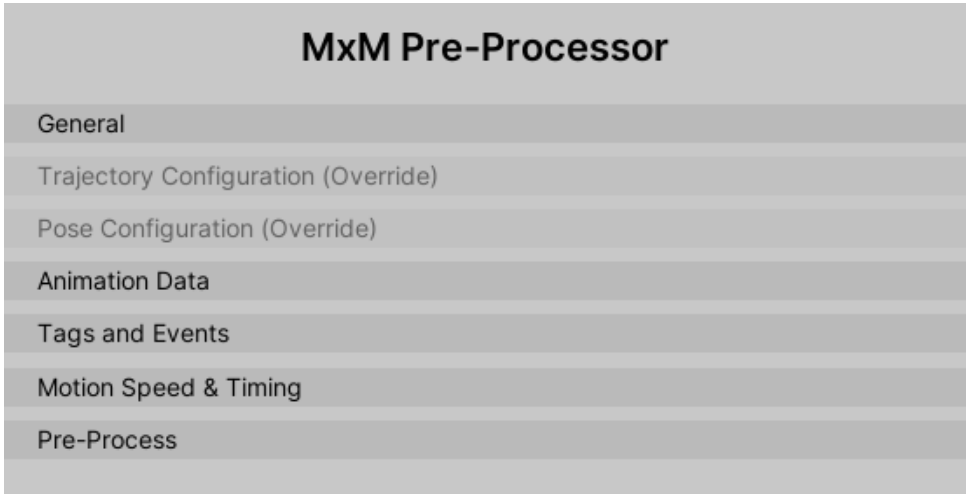


Figura 2.2: MxM PreProcessor

Di seguito spieghiamo le varie sezioni del componente.

- In **General** vengono definite le due proprietà dell'agente virtuale corrispondente: game object di riferimento e l'intervallo di posa che definisce l'intervallo di tempo fra ogni posa definita.
- In **Trajectory Configuration** vengono definiti i punti di traiettoria che servono per creare la possibile traiettoria. Questi punti possono avere anche valori negativi, nel caso in cui si voglia considerare anche la traiettoria del passato nel calcolo, cosa molto conveniente perché avendo anche informazioni temporali del passato è più facile riuscire a calcolare una possibile traiettoria futura.
- In **Pose Configuration** vengono inserite le parti di posa (parti del corpo) delle quali abbiamo bisogno per effettuare il match ed il calcolo delle possibile posa futura.
- In **Animation Data** vengono inserite le varie animazioni (o i moduli di animazioni) suddivise in tre parti: Composites (le animazioni composte), Idle Sets (il set di idle animations) e i Blend Spaces.
- In **Tags and Events** vengono inseriti gli eventi ed i tag, che saranno spiegati più approfonditamente nella sezione successiva.

- In **Motion Speed e Timing** vengono inserite le informazioni di velocità e tempo del moto definiti all'interno di un Timing Preset.
- In **Pre-Process** è presente una spunta per definire se si vogliano generare clip modificate in base alla posa ed il tasto per iniziare il preprocess.

2.1.2 Animazioni

Le animazioni che vengono utilizzate sono delle semplici clip di animazione senza limiti di creazione di essa. Può essere effettuata una pre-elaborazione precedente all'inserimento nell'MxM Pre-Process per fare in modo che il sistema sia facilitato nel calcolo della posa. L'elaborazione è diversa in base al tipo di animazione che stiamo utilizzando: nelle composites si possono inserire delle possibili animazioni antecedenti o successive in modo che sia più facile fare il match delle pose fra animazioni possibilmente legate fra loro, nelle idle animations, invece, è possibile definire delle animazioni di transizione che poi possano portare all'idle oppure delle animazioni secondarie che, casualmente, possono essere effettuate dall'agente virtuale.

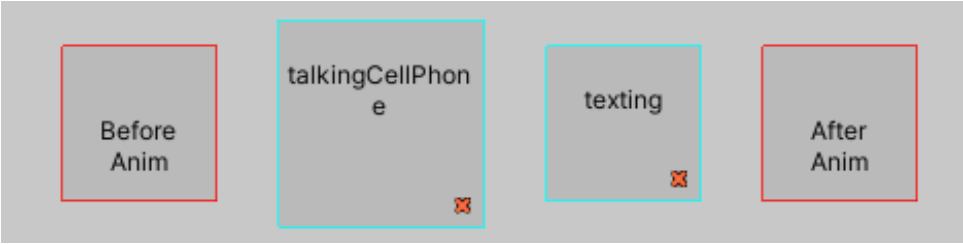


Figura 2.3: Composite Animation

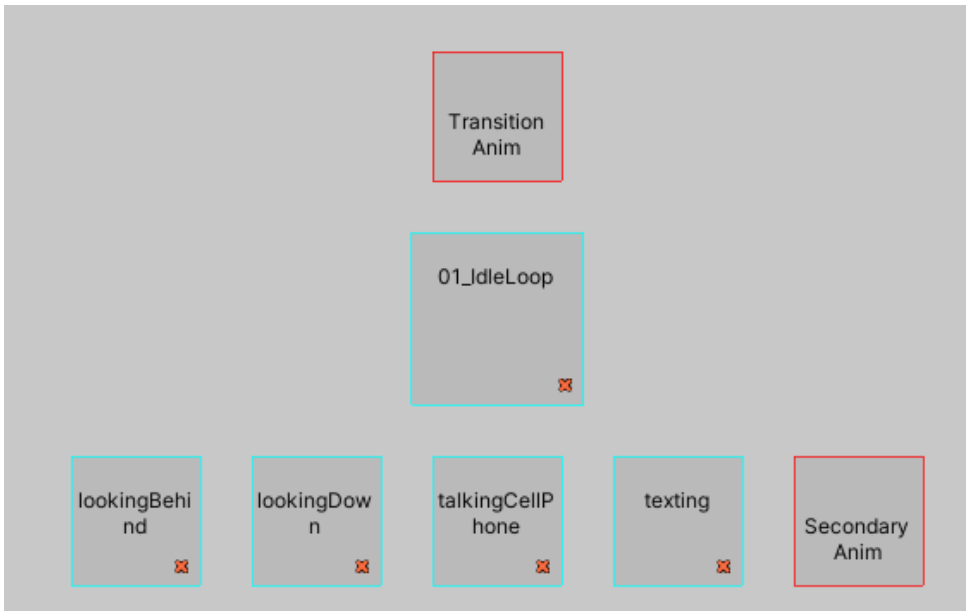


Figura 2.4: Idle animation

Nella prima immagine l'animazione principale è talkingCellPhone, mentre nella seconda è 01 IdleLoop.
Altre informazioni aggiuntive che si possono aggiungere alle animazioni riguardano i **tag**, che possono essere di due tipi Require e Favour. I tag servono per definire una o più animazioni per poi essere facilmente chiamate da codice attraverso il settaggio del tag principale. Tramite le funzioni preimpostate come "SetRequiredTag" si può definire il gruppo di animazioni che possono fare il match con le pose dell'avatar, così limitando le animazioni solo ad un gruppo ristretto. Gli **eventi** sono delle animazioni specifiche che possono essere triggerate in determinate condizioni (la condizione può essere scelta a piacere). Queste animazioni che possono essere triggerate hanno come tag "DoNotUse" poiché possono essere svolte solo ed esclusivamente quando viene triggerato l'evento. Per far in modo che l'animazione sia triggerata in maniera opportuna bisogna andare a definire delle parti di azione nella timeline, come si può vedere nell'immagine.

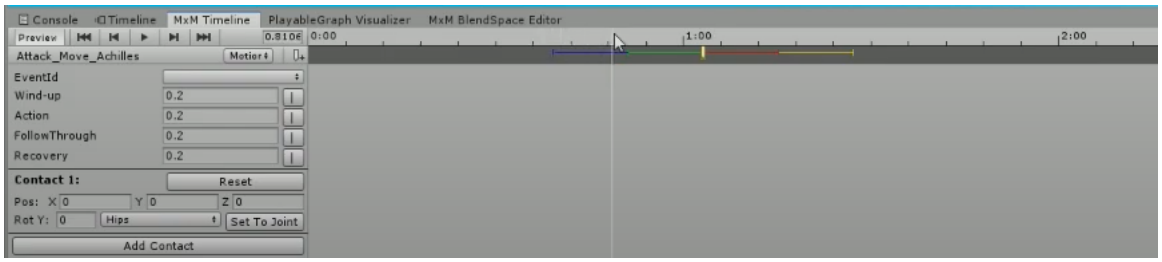


Figura 2.5: Timeline di MxM Event

Una volta definito l'ID dell'evento che servirà per la parte di programmazione, bisogna settare sulla timeline le parti di "wind-up", "action", "follow trough" e "recovery" dell'animazione.
Nel caso in cui ci siano dei contatti, e che quindi l'avatar abbia un contatto fisico con parti di stanza o oggetti, devono essere definiti nell'apposita sezione "Contact", all'interno della quale si definiscono le coordinate locali di contatto, la rotazione su Y e l'osso che andrà a contatto.

2.2 Final IK

Final IK è un tool per la *cinematica inversa* in Unity.
Una figura animata è modellata con uno scheletro di segmenti rigidi collegati con giunti, chiamato catena cinematica. Le equazioni cinematiche della figura definiscono la relazione tra gli angoli di giunzione della figura e la sua posa o configurazione. Il problema dell'animazione cinematica in avanti utilizza le equazioni cinematiche per determinare la posa dati gli angoli del giunto. La cinematica inversa è il processo matematico di calcolo dei parametri articolari variabili necessari per posizionare l'estremità di una catena cinematica, come un manipolatore di robot o lo scheletro di un personaggio dell'animazione, in una data posizione e orientamento rispetto all'inizio catena. Dati i parametri del giunto, la posizione e l'orientamento dell'estremità della catena, ad es. la mano del personaggio o del robot, in genere può essere calcolata direttamente utilizzando più applicazioni di formule trigonometriche, un processo noto come cinematica diretta. Tuttavia, l'operazione inversa è, in generale, molto più impegnativa.

2.2.1 Aim IK

Il risolutore AimIK è una modifica dell'algoritmo CCD che ruota una gerarchia di ossa per fare in modo che una trasformazione figlio di quella gerarchia miri a un obiettivo. Si differenzia dalla funzionalità di base Animator.SetLookAtPosition, perché è in grado di puntare con precisione le trasformazioni non allineate all'asse principale della gerarchia.
AimIK può produrre un retargeting molto stabile e dall'aspetto naturale dell'animazione dei personaggi. Con AimIK siamo in grado di compensare una singola posa o animazione di mira in avanti per mirare a bersagli anche quasi dietro il personaggio. Sono nel singolo punto del Quaternione a 180 gradi di offset il risolutore non può sapere in che modo girare la spina dorsale. AimIK fornisce una proprietà Clamp Weight per evitare problemi con quel problema di singolarità.
AimIK funziona anche con i limiti di rotazione, tuttavia è più soggetto a incepparsi rispetto ad altri solutori vincolati, se la catena è fortemente vincolata.

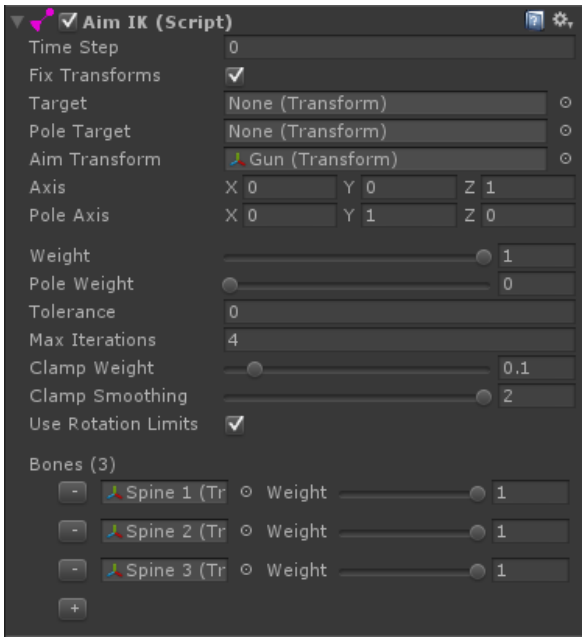


Figura 2.6: Componente AimIK

Sezioni all'interno del componente:

- il **Target** definisce la trasformata del target di riferimento. Se assegnato, IKPosition del risolutore verrà automaticamente impostato sulla posizione del target.
- Il **Pole Target** imposterà automaticamente polePosition alla posizione di questa trasformata.
- L'**Aim Transform** è la trasformata traguardo dell'IKPOsition.
- Con **Axis** si intende l'asse locale della Aim Transform che vuole raggiungere la IKPosition.
- Con **Pole Axis** si definisce l'asse locale della Aim Transform che si vuole mantenere orientato verso la Pole Position.
- Il **Pole Weight** è il peso che mantiene il Pole Axis di Aim Transform orientato verso la Pole Position.
- La **Tolerance** è l'offset minimo dall'ultimo angolo raggiunto. Smetterà di raggiungere l'angolo se la differenza dall'angolo raggiunto in precedenza è inferiore alla tolleranza.
- Le **Max Iterations** sono le massime iterazioni per frame. Se la tolleranza è zero, itera sempre fino a al numero settato.

- Il **Clamp Weight** è la rotazione di bloccaggio del risolutore. 0 è la rotazione libera, 1 è completamente bloccata.
- Il **Clamp Smoothing** è il numero di iterazioni di livellamento sinusoidale applicate al bloccaggio per renderlo più uniforme.
- Con **Bones** si definiscono le ossa usate dal risolutore per orientare la Aim Transform verso il bersaglio. Tutte le ossa devono essere parenti diretti della Aim Transform e ordinate in ordine decrescente. Si possono saltare le ossa nella gerarchia ma la gerarchia non può essere ramificata, il che significa che non si possono valutare le ossa da entrambe le mani. Il peso osseo determina la forza con cui viene utilizzato nel fare il bending nella gerarchia.

2.2.2 Full Body Biped IK

Il risolutore Full Body Biped IK (FBIK) viene utilizzato per avatar bipedi.
FBIK mappa qualsiasi personaggio bipede su un rig IK multi-effettori a bassa risoluzione, lo adatta e mappa il risultato sul personaggio. Questo viene fatto ogni fotogramma in LateUpdate, dopo che Mecanim/Legacy ha terminato l'animazione, quindi è completamente indipendente dal sistema di animazione.
Nelle sottosezioni successive vengono spiegate le sue proprietà.

Chains

Internamente, ogni arto e il corpo sono istanze della classe FBIKChain. La catena radice è il corpo, costituito da un singolo nodo, e gli arti sono i suoi figli. Questa configurazione forma l'albero IK multi-effettori attorno al nodo radice.

Nodes

I nodi sono membri delle catene. Ad esempio, una catena del braccio contiene tre nodi: braccio, avambraccio e mano. Ogni nodo mantiene un riferimento al suo osso (node.transform). Quando il risolutore sta elaborando o ha terminato, la posizione risolta dell'osso viene memorizzata in node.solverPosition.

Effettori

FBIK ha tre tipi di effettori: effettori finali (mani e piedi), effettori a metà corpo (spalle e cosce) e multi-effettori (corpo). Gli end-effector possono essere ruotati mentre la modifica della rotazione del mid-body e i multi-effector non hanno alcun effetto. La modifica della rotazione dell'effettore finale modifica anche la direzione di piegatura dell'arto (a meno che non si utilizzino obiettivi di piegatura per

ignorarlo). L'effettore del corpo è un effettore multiplo, il che significa che trascina anche entrambi gli effettori della coscia (per semplificare il posizionamento del corpo). Gli effettori hanno anche la proprietà positionOffset che può essere utilizzata per manipolare molto facilmente con l'animazione sottostante. Gli effettori reimposteranno il loro positionOffset su Vector3.zero dopo ogni aggiornamento del risolutore.

Pulling, Reaching and Pushing

Ogni catena ha la proprietà "pull". Quando tutte le catene hanno un pull uguale a 1, il peso del pull viene distribuito equamente tra gli arti. Ciò significa che il raggiungimento di tutti gli effettori non è garantito se sono molto lontani l'uno dall'altro. Il risultato può essere modificato o migliorato modificando il parametro "reach" della catena, aumentando il numero di iterazioni del risolutore o aggiornando il risolutore più di una volta per fotogramma. Tuttavia, quando ad esempio la catena del braccio sinistro ha un peso di pull pari a 1 e tutte le altre hanno 0, si può trascinare il personaggio dalla sua mano sinistra a infinito senza perdere il contatto. I valori Push e Push Parent determinano quanto un arto trasferisce energia ai suoi nodi genitori quando il target è a portata.

Mapping

IKSolverFullBodyBiped crea un'armatura ad alta velocità a risoluzione molto bassa. L'avatar, probabilmente, ha molte più ossa nella sua spina dorsale, potrebbe avere ossa tortuose nelle braccia e nelle ossa delle spalle o dell'anca e così via. Pertanto, il risolutore deve mappare lo scheletro ad alta risoluzione allo scheletro del risolutore a bassa risoluzione prima di adattarlo e viceversa dopo che il risolutore ha terminato. Esistono 3 tipi di mappatori: IKMappingSpine per mappare il bacino e la colonna vertebrale, IKMappingLimb per gli arti (compresa la clavicola) e IKMappingBone per la testa.

2.2.3 Sistema di interazione

Per effettuare delle interazioni all'interno del mondo virtuale vi sono diversi componenti che devono essere attaccati sia agli oggetti interagibili sia all'agente che deve interagire. Oltre ai componenti che verranno descritti nelle sotto sezioni è fondamentale che l'agente interagente abbia il componente di FBIK.

Interaction System

Questo componente sarà attaccato all'avatar.

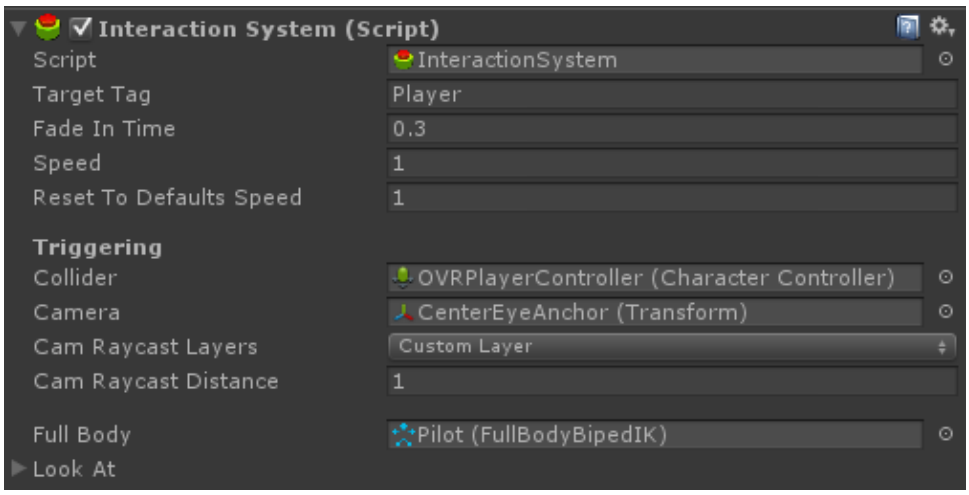


Figura 2.7: Componente Interaction System

Sezioni all'interno del componente:

- Il **Target Tag** identifica il tag che devono avere i target per essere utilizzati da questo sistema di interazione. Può essere anche vuoto, nel caso in cui non si voglia definire un gruppo di oggetti ben definito.
- Il **Fade In Time** indica il tempo di dissolvenza in tutti i canali utilizzati dall'interazione (weight of the effector, reach, pull..)
- Con **Speed** si definisce la velocità di tutte le interazioni dell'avatar.
- Il **Reset To Defaults Speed** se maggiore di zero, riporta tutti i canali FBBIK utilizzati dal sistema di interazione ai valori predefiniti o iniziali quando non sono in interazione.
- Il **Collider** è il collisore (componente Collider di Unity) che registra gli eventi OnTriggerEnter e OnTriggerExit con InteractionTriggers.
- La **Camera** verrà utilizzato dai trigger di interazione che richiedono la posizione della telecamera. Assegna la visuale in prima persona del personaggio.
- I **Cam Raycast Layers** sono tutti i layers che verranno trasmessi dalla fotocamera. Dovrebbero essere inclusi tutti i trigger di interazione che esaminano i collider target.

- La **Cam Raycast Distance** è la massima distanza di raycasting che si può avere dalla camera.
- Il **Full Body** è il componente FBIK dell'avatar.
- Il **Look At** è il componente Look At dell'avatar.

Interaction Object

Questo componente dovrebbe essere aggiunto agli oggetti di gioco con cui desideriamo interagire. Contiene la maggior parte delle informazioni sulla natura delle interazioni. Non specifica quali parti del corpo verranno utilizzate, ma piuttosto l'aspetto grafico e l'animazione dell'interazione. In questo modo le caratteristiche di un'interazione sono definite dall'oggetto e possono essere condivise tra più effettori.

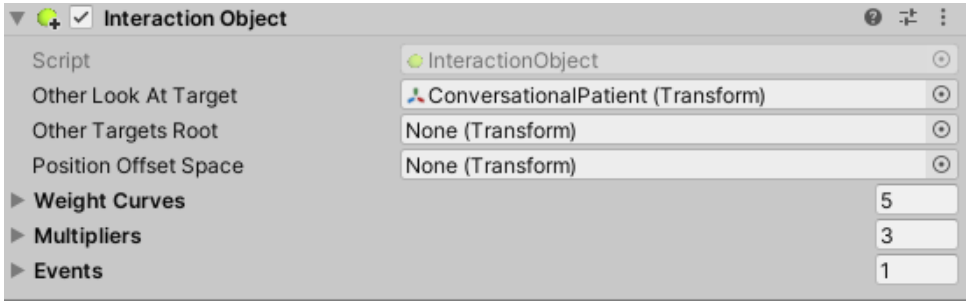


Figura 2.8: Componente Interaction Object

Sezioni all'interno del componente:

- **Other Look At Target** indica il target da guardare, può essere utilizzato solo nel caso in cui il target abbia InteractionLookAt come componente.
- **Other Targets Root** è la root Transform dell'Interaction Target, viene utilizzato per trovare automaticamente tutti i componenti Interaction Target.
- Il **Position Offset Space** se assegnato, tutti i canali PositionOffset verranno applicati nello spazio di rotazione di questa trasformazione. In caso contrario, saranno nello spazio di rotazione del personaggio.
- Le **Weight Curves** definiscono il processo dell'interazione. L'interazione sarà lunga quanto la curva di peso più lunga nell'elenco. Il valore orizzontale della curva rappresenta il tempo trascorso dall'inizio dell'interazione. Il valore verticale rappresenta il peso del suo canale. Possono essere di vario tipo: PositionWeight , RotationWeight, PositionOffsetX, PositionOffsetY, PositionOffsetZ, Pull, Reach, RotateBoneWeight, Push, PushParent, PoserWeight.

- I **Multipliers** sono progettati per ridurre la quantità di lavoro con AnimationCurves. Se si dovesse avere bisogno che la curva di rotazioneWeight sia la stessa della curva positionWeight, si potrebbe usare un multipier invece di duplicare la curva.
- Gli **Events** possono essere utilizzati per attivare animazioni, messaggi, pause di interazione o riprese in un determinato momento dall'inizio dell'interazione.

Interaction Target

Se l'oggetto di interazione non ha target di interazione, la posizione e la rotazione dell'oggetto di interazione stesso verranno utilizzate per tutti gli effettori come target. Tuttavia, se si dovesse avere bisogno di posare una mano in modo molto preciso, si dovrà creare un Interaction Target.

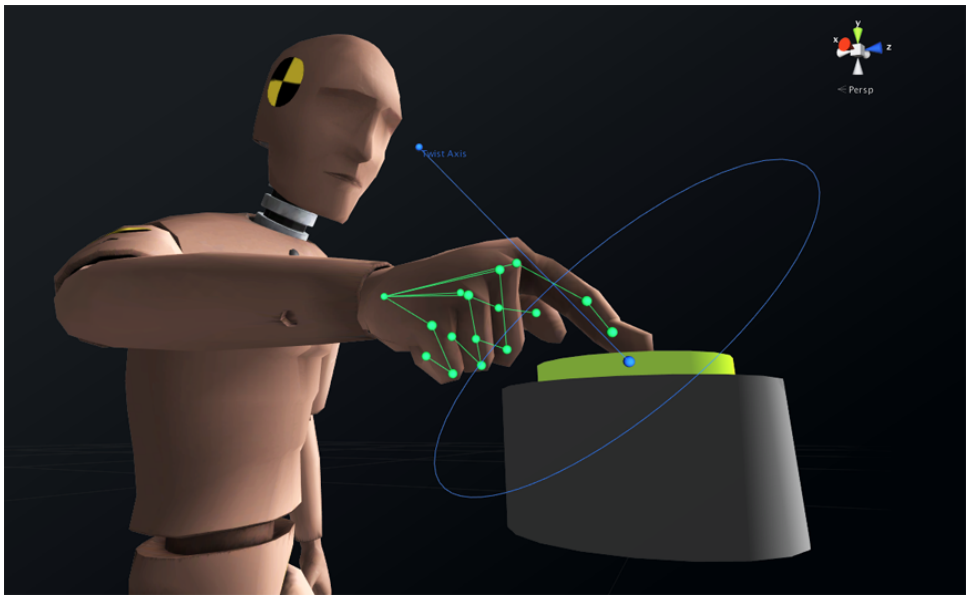


Figura 2.9: Esempio di utilizzo di un Interaction Target

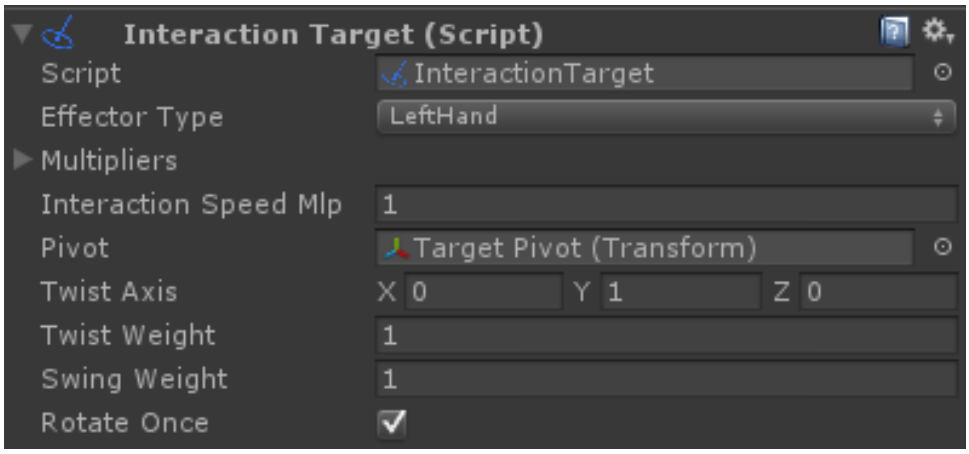


Figura 2.10: Componente Interaction Target

Sezioni all'interno del componente:

- **Effector Type** indica il tipo di effettore FBBIK a cui corrisponde questo target.
- Con **Multipliers** si intendono i moltiplicatori della curva del peso che consentono di sovrascrivere i valori della curva del peso per diversi effettori.
- L' **Interaction Speed Mlp** consente di modificare la velocità dell'interazione per diversi effettori.
- Il **Pivot** definisce il perno su cui ruotare/oscillare l'Interaction Target.
- Il **Twist Axis** è l'asse di torsione del bersaglio di interazione.
- Il **Twist Weight** è il peso della torsione del bersaglio dell'interazione verso l'osso effettore all'inizio dell'interazione.
- Lo **Swing Weight** è il peso dell'oscillazione del bersaglio dell'interazione verso l'osso effettore all'inizio dell'interazione. Ciò farà sì che la direzione dal perno all'InteractionTarget corrisponda alla direzione dal perno all'osso effettore.
- Se **Rotate Once** sarà flaggato allora girerà/oscillerà attorno al perno solo una volta all'inizio dell'interazione.

2.3 SALSA LipSync

SALSA LipSync è un sistema in tempo reale per creare la sincronizzazione labiale dall'ingresso audio senza la necessità di mappatura o baking dei fonemi.Essendo una soluzione in tempo reale offre opportunità che non possono essere effettuate tramite soluzioni di mappatura dei fonemi, come: input microfono e input da testo a voce. In tempo reale significa che non è necessario eseguire l'analisi della mappatura dei fonemi.
Per essere utilizzato basta semplicemente aggiungere due componenti ad un agente virtuale che abbia dei blend shapes:

- **SALSA**: è il componente principale e contiene molte informazioni fondamentali per il lip sync.

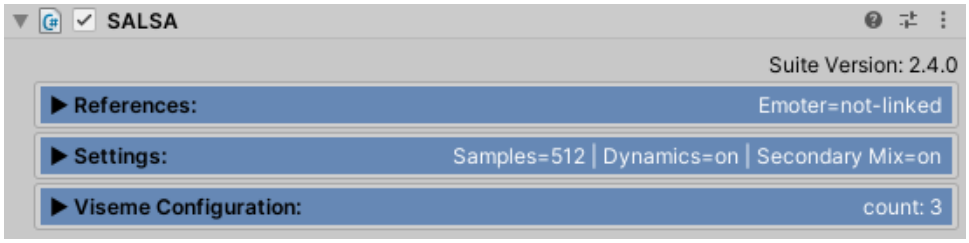


Figura 2.11: Componente SALSA

All'interno di References bisogna inserire l'audio source di riferimento, il componente Queue Processor e, nel caso in cui dovessero servire, delle Emoter, che servono a gestire la parte emozionale dell'espressione.

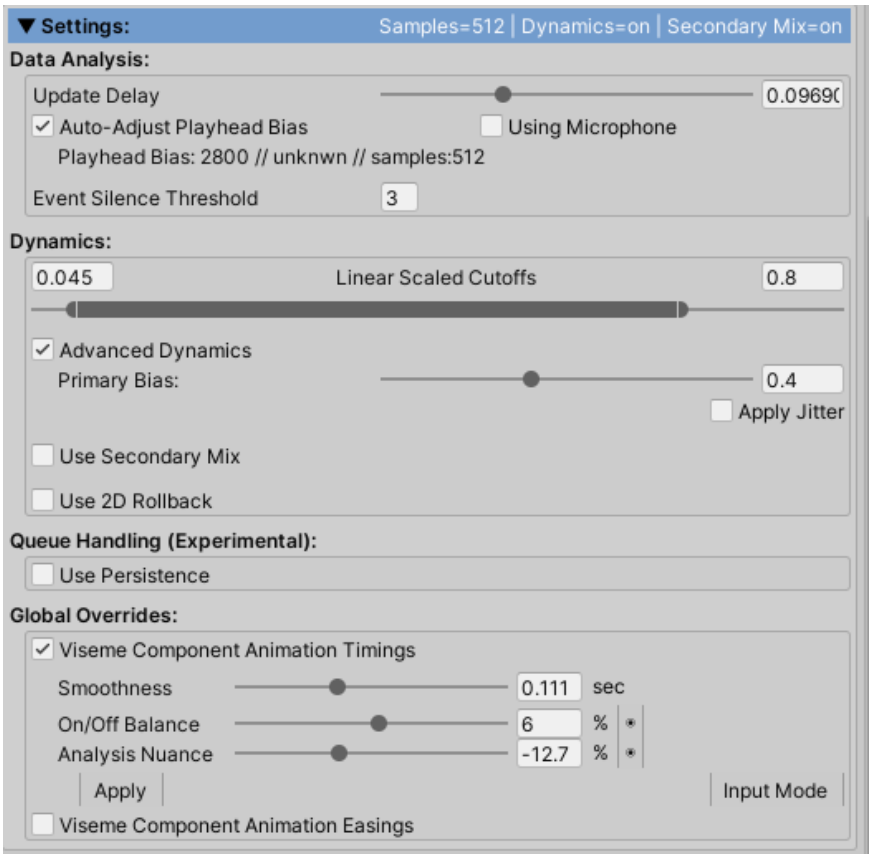


Figura 2.12: Settings SALSA

La parte delle impostazioni è più complessa e racchiude le informazioni principali che servono alla creazione del lip sync.

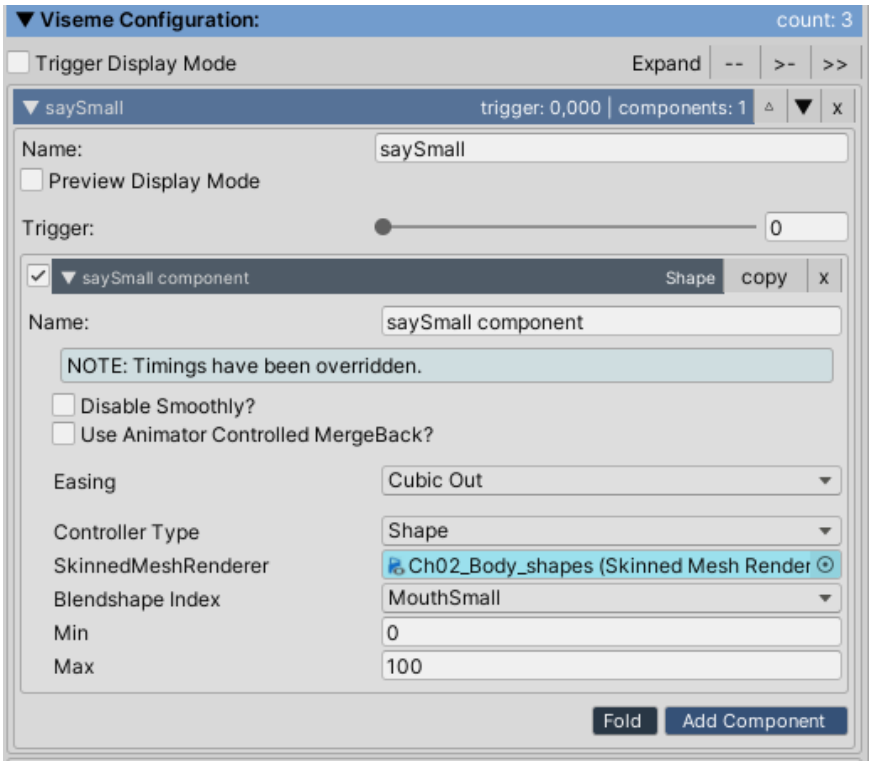


Figura 2.13: Esempio di configurazione di uno dei tre visemi utilizzati

All'interno di questa parte del componente bisogna specificare quante tipologie di visemi si vogliano utilizzare e definire il tipo di blend shape creato precedentemente per quel preciso visema. Durante la creazione del lip sync il componente riuscirà a fare il blending dei vari visemi adattandoli all'audio uscente.

- **Queue Processor** può essere considerato il controllore del traffico aereo per la suite SALSA LipSync. Tiene traccia delle animazioni di attivazione e disattivazione e risolve eventuali conflitti per evitare conflitti su ciò che un'animazione dovrebbe fare.

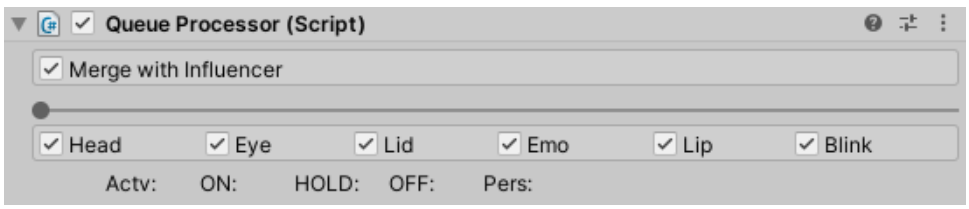


Figura 2.14: Componente Queue Processor

2.4 Tacotron

Tacotron è un modello di rete neurale che ha lo scopo di sintetizzare la voce da un testo (sistema di Text To Speech). Questa tipologia di rete neurale ha bisogno di uno sforzo notevole poiché le reti sono formate da diversi componenti con scopi e limitazioni diverse. I componenti che ogni TTS dovrebbe avere sono: un frontend di testo che estragga varie caratteristiche linguistiche, un modello di durata, un modello di previsione delle caratteristiche acustiche e un complesso vocoder basato sull'elaborazione del segnale.

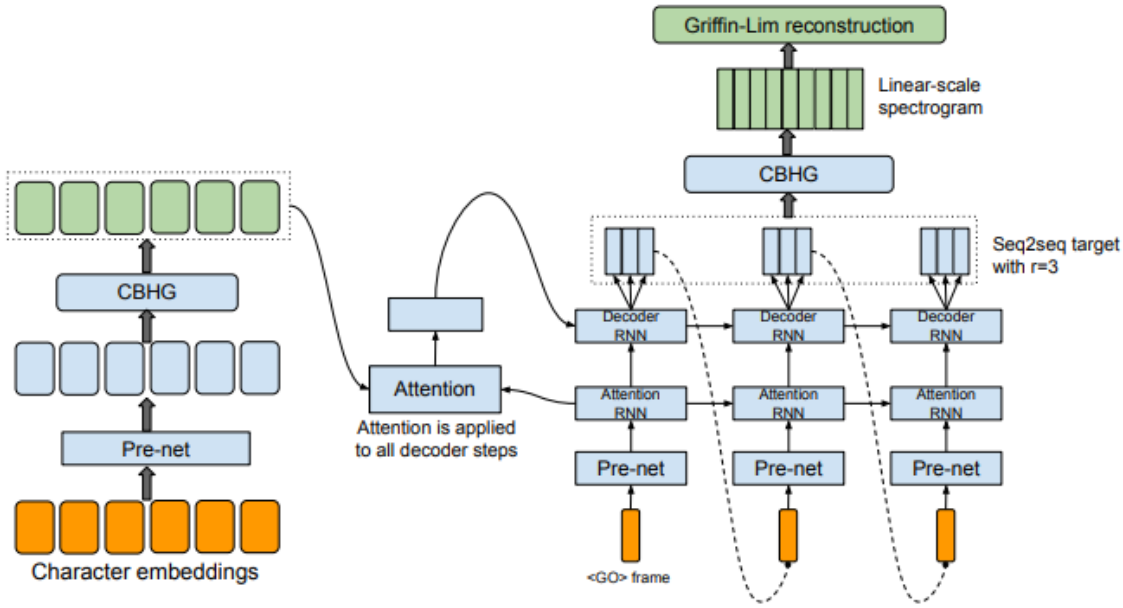


Figura 2.15: Modello di rete Tacotron

Il modello accetta i caratteri come input ed emette i corrispondenti spettrogrammi grezzi, che vengono inviati all'algoritmo di ricostruzione Griffin-Lim per sintetizzare il parlato.

2.4.1 CBHG

CBHG consiste in un banco di filtri convoluzionali 1-D, seguiti da highway networks e un'unità ricorrente gated bidirezionale (GRU) rete neurale ricorrente (RNN). CBHG è un potente modulo per estrarre rappresentazioni da sequenze. La sequenza di input viene prima convoluta con K insiemi di filtri convoluzionali 1-D, dove il k-esimo insieme contiene Ck filtri di larghezza k (cioè k = 1, 2, . . . , K). Questi filtri modellano esplicitamente le informazioni locali e contestuali. Le uscite

di convoluzione sono impilate insieme e ulteriormente raggruppate nel tempo per aumentare le invarianze locali. Viene utilizzato il passo di 1 per preservare la risoluzione temporale originale e, successivamente, la sequenza elaborata viene fatta passare attraverso alcune convoluzioni 1-D a larghezza fissa, i cui output vengono aggiunti alla sequenza di input originale tramite connessioni residue. Il batch viene normalizzato in tutti i livelli convoluzionali. Una volta ottenuti gli output della rete convoluzionale vengono, inizialmente, immessi all'interno di una highway network multistrato per l'estrazione di caratteristiche di alto livello, e poi successivamente attraverso una GRU RNN bidirezionale per estrarre le funzionalità sequenziali dal contesto sia in avanti che indietro.

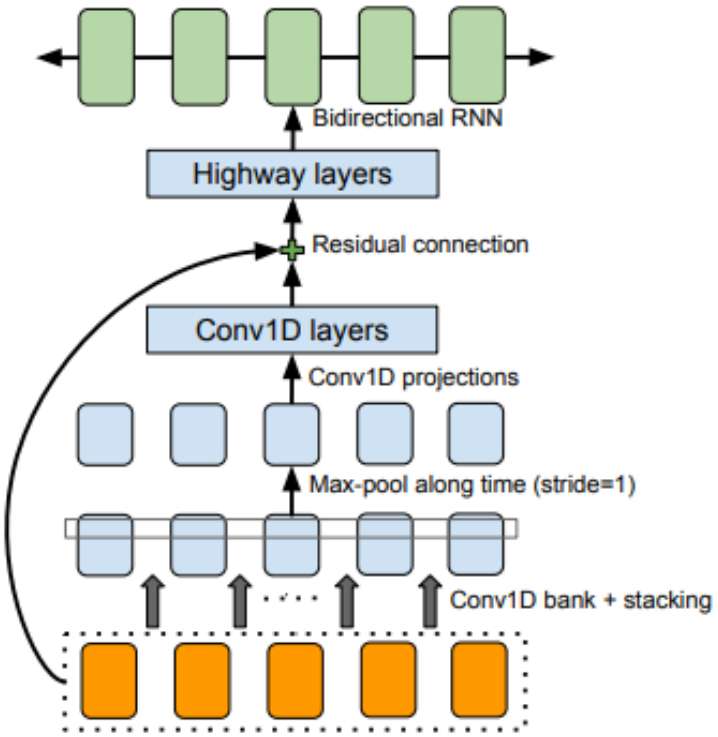


Figura 2.16: Modulo CBHG

2.4.2 Codificatore

Il codificatore ha il compito di estrarre rappresentazioni sequenziali robuste del testo. Il codificatore ha come input una sequenza di caratteri, e, ognuno di questi, è rappresentato come un vettore one-hot (quindi, un vettore all'interno del quale tutti i bit sono posti a 0 tranne uno) e poi incorporato in un vettore continuo.

Una volta ottenuto il vettore continuo vengono applicate delle trasformazioni lineari, complessivamente chiamate "Pre-net", utilizzando questo layer come collo di bottiglia con dropout in modo da migliorare la convergenza e la generalizzazione.

2.4.3 Decodificatore

Il decodificatore di attenzione è content-based basato su tanh, in cui uno stateful recurrent layer produce le query di attenzione in ogni fase temporale (time step) del decodificatore. Il vettore di contesto e l'output della cella di attenzione RNN vengono concatenati per andare a formare l'input agli RNN del decodificatore. Per accelerare la convergenza viene utilizzato uno stack di GRU con connessione residue verticali.
Una delle parti fondamentali è la scelta del **target** poiché, nonostante si possa prevedere direttamente lo spettrogramma grezzo, questa rappresentazione risulta ridondante nell'apprendere l'allineamento tra segnale vocale e testo. Vista questa ridondanza, vengono utilizzati target diversi per la decodifica seq2seq e la sintesi della forma d'onda. Il target seq2seq può essere altamente compresso purché fornisca sufficiente intelligibilità e informazioni sulla prosodia per un processo di inversione, che potrebbe essere riparato o addestrato. Dato questo, viene utilizzato lo spettrogramma in scala mel a 80 bande, ma potrebbe essere utilizzato anche altro, come il cepstrum che è un target con meno bande e più conciso. Successivamente i dati passano attraverso una rete di post-elaborazione per convertire il target seq2seq in una forma d'onda. Per finire, viene utilizzato un semplice livello di output fully-connected per ottenere le predizioni fuoriuscenti dal decodificatore.

2.4.4 Rete di post-processing e sintesi dell'onda d'audio

Il compito della rete di post-processing è quello di convertire il target seq2seq in un target che può essere sintetizzato in forme d'onda. Come sintetizzatore viene utilizzato Griffin-Lim così che la rete di post-processing riesca ad imparare a prevedere la grandezza spettrale campionata su una scala di frequenza lineare e può vedere interamente la sequenza codificata. Come modulo per la rete viene utilizzato un CBHG. Per ridurre gli artefatti bisogna aumentare le magnitudine di circa 1.2 di potenza prima di passare attraverso Griffin-Lim, questo dovrebbe essere dovuto all'effetto dell'algoritmo di miglioramento delle armoniche. La convergenza si ha dopo circa 50 iterazioni.

2.5 Azure Cognitive Services

Il servizio di Microsoft Azure è una piattaforma di cloud computing. All'interno di questa piattaforma si possono trovare molte tipologie di servizi, noi abbiamo utilizzato due dei tools all'interno di essa: **Speech To Text** e **LUIS**. Questi servizi sono facilmente integrabili in Unity attraverso una SDK.
Per effettuare le chiamate per i vari servizi è disponibile un tipo di sottoscrizione per studenti gratuita, che, in maniera diversa per ogni applicazione di Azure, ingloba un numero di chiamate all'applicazione (ad esempio per il servizio di LUIS è previsto un massimo di 5 chiamate al secondo, per la nostra applicazione ne facciamo un numero notevolmente inferiore).

2.5.1 Speech To Text

Il servizio Speech To Text di Azure consente di generare del testo una volta dato un determinato file audio. Il file audio che si passa all'applicazione tramite SDK deve essere in formato wav, poiché per ora non vengono supportati altri formati. Il servizio è attualmente in crescita, quindi, probabilmente, in un futuro prossimo anche altri formati saranno supportati.
L'audio può essere anche registrato in real time dal microfono e poi fare la chiamata al servizio di Azure; questo è il tipo di utilizzo più tipico del servizio ed è anche il modo in cui lo abbiamo utilizzato nei nostri use cases.
Il livello di accuratezza del sistema è notevolmente più alta per la lingua inglese rispetto alla lingua italiana: avendo due use case in lingua diversa abbiamo potuto testare la diversità di precisione e in italiano spesso capitava che facesse degli errori mentre in inglese, che non è la nostra lingua madre e parliamo con un accento abbastanza marcato, gli errori di scrittura del testo si riducevano notevolmente.

2.5.2 LUIS

Il servizio LUIS (Language Understanding Intelligent Service) è un'API che offre un servizio di intelligenza artificiale per la comprensione del linguaggio naturale (in inglese NLU, Natural Language Understanding).
Il servizio LUIS si basa sulla definizione di alcuni **intent** che racchiudono alcune frasi definite a priori. Quindi, ogni intent ha una serie di frasi già preimpostate che servono a identificarlo.
Tramite un servizio di Speech To Text viene effettuata la chiamata (query) a Luis che cercherà, tramite un servizio di Machine Learning, di capire a quale intent quella determinata frase appartenga. Una volta identificato l'intent di appartenenza l'API risponderà alla query dando come risposta l'intent e uno **score** che identifica un punteggio di stima e di attendibilità della risposta erogata dal servizio.

Capitolo 3

Sperimentazione relativa alla sintesi vocale

In questo capitolo verrà analizzata la sperimentazione e la ricerca relativa alla sintesi vocale. Ogni tipo di approccio ha in comune l'utilizzo del Machine Learning, ma con modelli e tipologie di implementazione diverse.
Tutte la varie sperimentazioni sono state fatte in inglese per la mancanza di dataset in italiano.

3.1 GST-Tacotron

GST-Tacotron [28] è un modello basato su Tacotron, un modello di rete neurale descritto nel capitolo precedente. Il modello di GST-Tacotron proposto è costituito da un codificatore di riferimento, style attention, style embedding ed il modello Tacotron.

3.1.1 Training

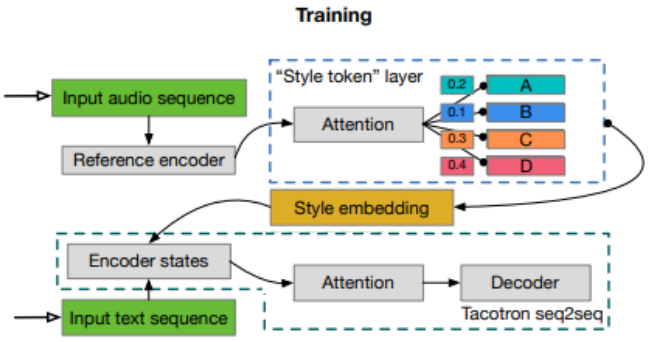


Figura 3.1: Training del GST-Tacotron

Durante la fase di **Training** l'encoder di riferimento comprime la prosodia di un segnale audio a lunghezza variabile in un vettore a lunghezza fissa, che chiamiamo embedding di riferimento. Durante l'allenamento, il segnale di riferimento è l'audio ground-truth.
L'embedding del riferimento viene passato a un style token layer, dove viene utilizzato come vettore di query a un modulo di attenzione. Qui, l'attenzione apprende una misura di somiglianza tra l'embedding di riferimento e ciascun token in un banco di embeddings inizializzati casualmente. Questo insieme di embeddings, che alternativamente chiamiamo global style token, GST o embeddings di token, è condiviso in tutte le sequenze di addestramento.
Il modulo di attenzione emette una serie di pesi di combinazione che rappresentano il contributo di ogni tyle token all'incorporamento di riferimento codificato. La somma pesata dei GST, che chiamiamo style embeddings, viene passata al codificatore di testo per il condizionamento ad ogni time step.
Il style token layer viene addestrato congiuntamente con il resto del modello, guidato solo dalla loss di ricostruzione dal decoder Tacotron. I GST quindi non richiedono alcuno stile esplicito o etichette di prosodia.

3.1.2 Esperimenti

L'implementazione di GST-Tacotron di syang1993 si basa su Tensorflow con pre-allenamenti in lingua coreana. Abbiamo utilizzato come dataset sia il Blizzard 2013 [10] sia LJSpeech [8], inizialmente in maniera separata, poi utilizzandoli entrambi contemporaneamente dopo aver effettuato il pre-process separatamente. L'allenamento finale è durato cinque giorni arrivando a 5000 step, ma senza ottenere dei risultati decenti. L'audio non risultava intellegibile. Informandoci meglio all'interno degli issues e nei forum abbiamo notato che per ottenere dei buoni risultati si dovessero superare almeno i 100000 step, richiedendo del tempo necessario non accettabile per il nostro lavoro non avendo la disponibilità di modelli pre-allenati in inglese.
Un'altra implementazione è quella di KinglittleQ che, invece, si basa su Pytorch e sempre con allenamenti in linqua coreana. I risultati sono stati leggermente migliori rispetto a quelli di syang1993, riuscendo ad ottenere almeno una parola intellegibile, ma comunque non riuscendo a sintetizzare una frase in maniera completa. Anche in questo caso sarebbero serviti molti giorni di allenamento per poter ottenere dei risultati almeno intellegibili e la mancanza di pesi pre-allenati in lingua inglese ci ha fatto abbandonare questa possibilità.
La valutazione di abbandonare l'idea dell'utilizzo del GST-Tacotron è nata anche dal fatto che queste citate non siano le implementazioni ufficiali.

3.2 Tacotron 2

Il modello di rete neurale Tacotron 2 [21] è costituito da due componenti: una rete di predizione delle caratteristiche da seq2seq ricorrente con attenzione che predice una sequenza di frame dello spettrogramma mel da una sequenza di caratteri di input e una versione modificata di WaveNet che genera campioni di forme d'onda nel dominio del tempo condizionati dallo spettrogramma di mel predetto.
La rappresentazione che viene utilizzata per allenare i due componenti in maniera separata è una rappresentazione acustica di basso livello: lo spettrogramma di melfrequenza. Questa rappresentazione è più uniforme rispetto ai campioni di forme d'onda ed è più facile da addestrare utilizzando una perdita di errore al quadrato perchè è invariante rispetto alla fase all'interno di ciascun frame. Uno spettrogramma a frequenza Mel è correlato allo spettrogramma a frequenza lineare, ovvero l'ampiezza della trasformata di Fourier a breve tempo (STFT). Si ottiene applicando una trasformata non lineare all'asse delle frequenze dell'STFT, ispirata alle risposte misurate dal sistema uditivo umano, e riassume il contenuto in frequenza con meno dimensioni. L'uso di una tale scala di frequenza uditiva ha l'effetto di enfatizzare i dettagli nelle frequenze più basse, che sono fondamentali per l'intelligibilità del parlato, mentre de-enfatizza i dettagli ad alta frequenza, che sono dominati da fricative e altri scoppi di rumore e generalmente non hanno bisogno di essere modellati con alti fedeltà. Gli spettrogrammi lineari, invece, scartano le informazioni di fase (e sono quindi con perdite), algoritmi come Griffin-Lim sono in grado di stimare queste informazioni scartate, il che consente la conversione nel dominio del tempo tramite la trasformata di Fourier inversa a breve tempo. Gli spettrogrammi di Mel scartano ancora più informazioni, presentando un impegnativo problema inverso. Tuttavia, rispetto alle caratteristiche linguistiche e acustiche utilizzate in WaveNet, lo spettrogramma Mel è una rappresentazione acustica più semplice e di livello inferiore dei segnali audio. Dovrebbe quindi essere semplice per un modello WaveNet simile, condizionato su spettrogrammi mel, generare audio, essenzialmente come un vocoder neurale.

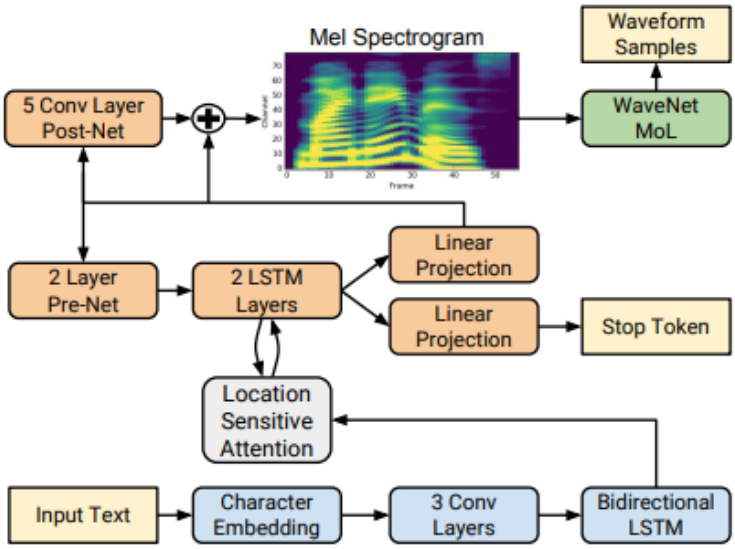


Figura 3.2: Modello Tacotron 2

Come in Tacotron, gli spettrogrammi mel sono calcolati attraverso una trasformata di Fourier di breve durata (STFT).
Si trasforma la magnitudine STFT nella scala mel utilizzando un banco di filtri mel a 80 canali che copre 125 Hz a 7,6 kHz, seguito dalla compressione della gamma dinamica log. Prima della compressione del registro, le grandezze di uscita del banco di filtri vengono ritagliate a un valore minimo di 0,01 per limitare l'intervallo dinamico nel dominio logaritmico.
La rete è composta da un encoder e un decoder con attenzione. Il codificatore converte una sequenza di caratteri in una rappresentazione di caratteristiche nascoste che il decodificatore utilizza per prevedere uno spettrogramma. caratteri di input sono rappresentati utilizzando un'incorporazione di caratteri a 512 dimensioni appresa, che vengono fatti passare attraverso una pila di 3 strati convoluzionali ciascuno contenente 512 filtri con forma 5 x 1, ovvero, dove ogni filtro si estende su 5 caratteri, seguito dalla normalizzazione batch e dalle attivazioni ReLU. Come in Tacotron, questi strati convoluzionali modellano il contesto a lungo termine nella sequenza di caratteri di input. L'output dello strato convoluzionale finale viene passato in un singolo strato bidirezionale LSTM contenente 512 unità (256 in ciascuna direzione) per generare le caratteristiche codificate. L'uscita dell'encoder viene data in pasto ad una rete di attenzione che riassume l'intera sequenza codificata come vettore di contesto a lunghezza fissa per ogni fase di output del decodificatore. Le probabilità di attenzione vengono calcolate dopo aver proiettato l'input e le caratteristiche di posizione su rappresentazioni nascoste a 128 dimensioni. Le caratteristiche della posizione vengono calcolate utilizzando 32 filtri di convoluzione 1-D di lunghezza 31.

La concatenazione dell'output LSTM e del vettore del contesto di attenzione viene proiettata attraverso una trasformata lineare per prevedere il frame dello spettrogramma target. Infine, lo spettrogramma mel previsto viene fatto passare attraverso una post-rete convoluzionale a 5 strati che prevede un residuo da aggiungere alla previsione per migliorare la ricostruzione complessiva. L'errore quadratico medio sommato (MSE) viene ridotto al minimo sia prima che dopo la post-rete per favorire la convergenza. Parallelamente alla predizione del frame dello spettrogramma, la concatenazione dell'output LSTM del decodificatore e il contesto di attenzione viene proiettato su uno scalare e fatto passare attraverso un'attivazione sigmoide per prevedere la probabilità che la sequenza di output sia stata completata. Questa previsione "stop token" viene utilizzata durante l'inferenza per consentire al modello di determinare dinamicamente quando terminare la generazione invece di generare sempre per una durata fissa.
Questo modello utilizza blocchi di costruzione più semplici rispetto a quelli del Tacotron originale, utilizzando LSTM e livelli convoluzionali nell'encoder e nel decodificatore invece di stack CBHG e GRU ricorrenti.

3.2.1 Esperimenti

Abbiamo utilizzato l'implementazione di "coqui" [6] del TTS. L'idea principale era quella di effettuare un fine-tuning con dei dataset con etichette emozionali. Il dataset che abbiamo utilizzato per il fine-tuning è **EmoV-DB** [1], un database registrato in inglese ed in francese. Questo database contiene sia voci femminili che maschili e cinque diverse emozioni: Neutral, Amused, Angry, Sleepy, Disgust. Abbiamo effettuato un fine-tuning di un modello pre-allenato di Tacotron 2 con il database Amused, ma vista la relativa quantità di ore (circa due per ogni emozione) il risultato ottenuto dopo tre giorni di allenamento non è stato soddisfacente. La mancanza di un database emozionale abbastanza grande rende il fine-tuning non adatto alla sintesi vocale.
Visti i risultati non accettabili da essere definiti "emozionali" abbiamo optato per utilizzare il Tacotron 2 base implementato da coqui.

Capitolo 4

Implementazione

Lo scopo principale del progetto è quello di fornire gli strumenti per una gestione semplice ed efficace del comparto di animazione e di sintesi vocale degli E^2CA. Nel far questo è necessario porre particolare attenzione nella struttura del codice e nella modularità dei componenti stessi aggiunti al framework. La motivazione è che nel campo della realtà virtuale, e in questo caso in particolare nel campo degli agenti conversazionali, c'è un costante rinnovamento nelle tecnologie e le soluzioni che si possono adottare sono molteplici dunque è fondamentale avere una struttura sottostante che permetta la modifica o la sostituzione dei tool utilizzati in maniera semplice e veloce. Questo è molto importante perchè permette di avere una struttura concettuale intuibile e facilmente comprensibile che nasconde al suo interno tutte le componenti del tool utilizzate senza esserne minimamente dipendente; questo permette anche a una persona che non è un programmatore di comprenderne comunque la logica e quindi di capire in che modo possa essere utilizzabile il framework, cosa fondamentale quando si lavora in team multidisciplinari.
Lo stesso discorso vale ovviamente anche per la sintesi vocale, altro componente del framework che necessita modularità e la possibilità di sostituire facilmente i tools utilizzati in modo semplice e veloce.

4.1 Design

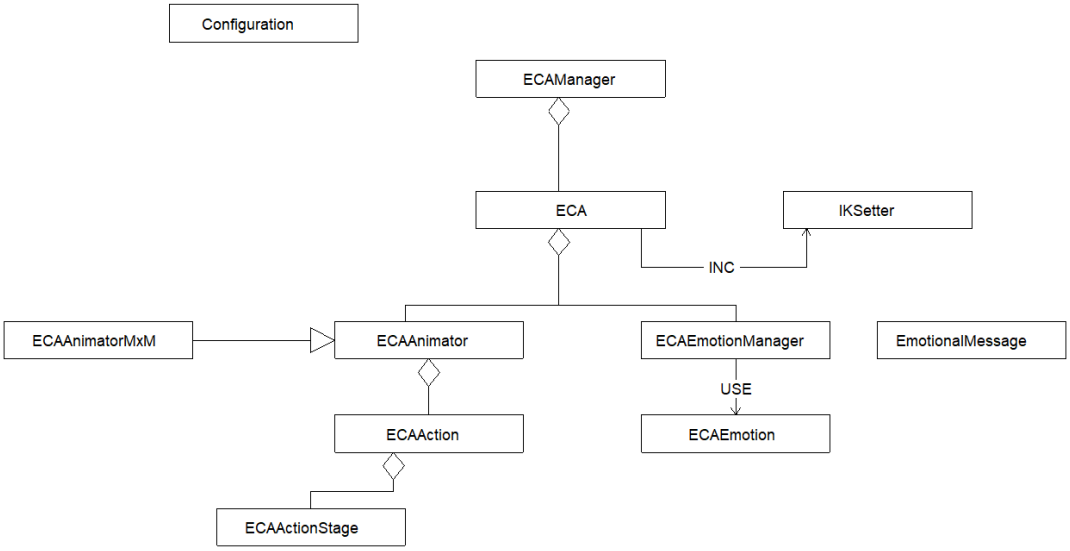


Figura 4.1: Design del framework

4.1.1 Gestione delle animazioni

L'idea di base è definire le animazioni dell'E^{2}CA attraverso le azioni che può compiere; in questo modo è possibile definire il comportamento dell'agente conversazionale attraverso una sequenza di azioni da effettuare. Le azioni sono a loro volta suddivise in sotto-azioni atomiche che sono responsabili dell'animazione nel suo complesso. É dunque fondamentale definire il concetto di ECAAction e ECAActionStage:

- **ECAAction:** è l'azione che l'E^{2}CA compie. Si compone normalmente di vari step; ad esempio, l'azione di sedersi prevde di arrivare davanti a una sedia, voltarsi e infine sedersi.

- **ECAActionStage:** sono azioni atomiche che compongono le ECAAction. Un ECAActionStage potrebbe essere ad esempio l'azione di raggiungere un punto nello spazio o prendere in mano un oggetto.

ECAAction

L'ECAAction è un'astrazione del concetto di azione: all'interno della classe non vengono lanciate direttamente le animazioni dell'agente ma serve piuttosto come

interfaccia tra la classe E^{2}CA e gli ECAActionStage e per la gestione di quest'ultimi rispetto agli eventi esterni che possono influenzare l'andamento dell'azione. Ogni azione infatti può essere messa in pausa, ripresa o direttamente abortita ed è possibile definire il comportamento specifico per questi eventi per ogni azione definita dal programmatore. In particolare, possono essere definite delle azioni non abortibili tramite il booleano **canAbort = true**, che quindi non potranno essere interrotte da eventi esterni e che raggiungeranno la propria fine senza interruzioni. Quando un azione viene interrotta o abortita lo segnala allo stage attuale che a sua volta termina o mette in pausa l'animazione. Le ECAAction possono essere viste infatti come una sequenza di ECAActionStage. Questo è molto utile in fase di programmazione perchè permette di definire degli stage generali (es. raggiungere un punto nello spazio) che possono successivamente essere utilizzati in più di un'azione.
Al suo interno dunque ogni ECAAction contiene una lista di ECAActionStage che viene scandita man mano che gli stage partono e vengono completati:

- Non appena viene chiamato il metodo **StartAction()**, viene fatto partire il primo stage della lista
- Quando uno stage finisce lo notifica all'azione che l'ha fatto partire e questa controlla se ci sono altri stage in coda; se sì fa partire il prossimo altrimenti segnala la fine dell'azione

La definizione delle ECAAction inoltre può avvenire in script a sè stanti o in qualsiasi altra parte del codice, il che permette di generare delle azioni on-demand quando se ne ha la necessità o nel caso in cui quell'azione non viene eseguita più volte.

```
public class ECAAction
{
    public ActionState State;
    public ECAActionStage ActualStage;
    public event EventHandler CompletedAction;
    protected bool canAbort;
    protected ActionName actionName;
    public ECAAnimator ecaAnimator;
    public ECAActionStage[] AllStages;

    //constructors
    public ECAAction(ECA eca, List<ECAActionStage> stages, bool canAbort =
        true) {...}
    public ECAAction(ECA eca, ECAActionStage stage, bool canAbort = true)
        {...}
    public ECAAction(ECA eca, bool canAbort = true) {...}

    //delegates for actual stage events
    protected virtual void OnStageFinished(object sender, EventArgs e) {...}
```

```
        protected virtual void OnStageAborted(object sender, EventArgs e) {...}
        protected virtual void OnStagePaused(object sender, EventArgs e) {...}

        //methods for action management
        public void Pause() {...}
        public void Abort() {...}
        public void Resume() {...}

        //methods for emotion management
        public virtual void OnEmotionChanged(ECAEmotion emotion) {...}
        public virtual void OnEmotionUpdated(ECAEmotion emotion) {...}

        public virtual void StartAction() {...}
        protected virtual void NextStage() {...}
        public virtual void OnCompletedAction() {...}
    }
```

Listing 4.1: Implementazione ECAAction.cs

ECAActionStage

L'ECAActionStage è il componente direttamente responsabile delle animazioni dell'agente: al suo interno vengono lanciati tutti i comandi che permettono alle varie animazioni di partire. La classe di base definisce dei metodi fondamentali di cui viene fatto l'override in tutte le classi derivate a seconda delle necessità e di quale sia lo scopo dello specifico stage:

- **StartStage()**
- **EndStage()**

É fondamentale definire inoltre i criteri che determinano la fine dello stage, altrimenti c'è il rischio di non terminare mai lo stage corrente e diventa impossibile dunque proseguire l'azione.

```
public abstract class ECAActionStage
{
    public event EventHandler StageFinished;
    public event EventHandler StageAborted;
    public event EventHandler StagePaused;

    public ActionState State;
    public IKSetter ikManager;
    protected ECAAnimator animator;
    protected bool waitStatus = false;
    protected double waitTime;
    protected DateTime startTime;
```

```
    //constructor
    public ECAActionStage(ECAAnimator ecaAnimator = null) { }

    //methods to check status of the stage
    public virtual void Update()
    public virtual void LateUpdate()
    protected void WaitFor(float seconds)
    protected virtual void OnEventComplete(object sender, EventArgs e)
    protected virtual void OnWaitCompleted()

    //methods for changing stage state
    public virtual void PauseStage()
    public virtual void ResumeStage()
    public virtual void AbortStage() { }
    public virtual void StartStage() { }
    public virtual void EndStage() { }

    //methods for react to external events
    public virtual void ReactToActionStart(object sender, EventArgs e)
    public virtual void ReactToStateUpdate(object sender, EventArgs e)
    public virtual void ReactToLabelUpdate(object sender, EventArgs e)
    public virtual void ReactToActionFinished(object sender, EventArgs e)
    public virtual void ReactToEmotionUpdated(ECAEmotion emotion) { }
    public virtual void ReactToEmotionChanged(ECAEmotion emotion) { }
}
```

Listing 4.2: Implementazione ECAActionStage.cs

ECACompositeAction e ECAParallelActionStage

Sono due tipi particolari di ECAAction e di ECAActionStage:

- **ECACompositeAction:** classe utilizzata per definire una macroazione con una logica interna più complessa di una semplice ECAAction. É composta da una lista di ECAAction e si possono definire le condizioni che determinano l'avanzamento a piacimento.

- **ECAParallelActionStage:** classe utilizzata se è necessario che più stage inizino in parallelo,dunque al suo interno si occupa di gestire l'inizio e la fine di 2 ECAActionStage. Quando viene chiamato il metodo *StartStage()* iniziano entrambi gli stage mentre il metodo *EndStage()* viene chiamato solo nel momento in cui entrambi gli stage sono finiti.

Utilizzo delle ECAAction

La gestione delle ECAAction di un E^2CA avviene attraverso l'utilizzo dell'**ECAAnimator**, dell'**IKSetter** corrispondente e di una **ECAActionList**.

ECA

La classe ECA si occupa di gestire la logica che porta l'agente a intraprendere un'azione o meno. Se vengono soddisfatte determinate condizioni definite dal programmatore può essere creata una ECAAction e messa in coda nella ECAActionList.

ECAAnimator

Fornisce tutti i metodi necessari agli ECAActionStage per far partire le varie animazioni e in generale nella gestione di qualsiasi tipo di movimento dell'agente. La classe ECAAnimator di base prevede l'utilizzo del Mechanim Animator di Unity ma nell' implementazione realizzata per questo progetto è stata creata una classe derivata da ECAAnimator che permettesse la gestione di un MxMAnimator, necessario per l'utilizzo del Motion Matching.

IKSetter

É la classe che si occupa di gestire tutti i componenti relativi all'inverse kinematic. Attraverso i metodi forniti è possibile modificare i target dei vari effector, nonchè i relativi pesi. Quando viene inizializzato, recupera autonomamente tutte le informazioni relative allo scheletro dell'avatar in modo da poter generare i vari componenti necessari; è possibile utilizzare un qualsiasi modello a patto che sia definito come umanoide all'interno di Unity. La classe è fondamentale nella gestione delle animazioni ed è utilizzata nei vari ECAActionStage perchè permette di adattare le animazioni all'ambiente circostante in maniera semplice e veloce.

ECAActionList

Si occupa di far iniziare e gestire la sequenza di azioni che l'E^2CA deve compiere. Nel momento in cui viene messa in coda un'azione, se questa è l'unica presente nella lista, allora inizia immediatamente, altrimenti si attende la fine dell'azione precedente e poi inizia. Attraverso questa classe è possibile quindi gestire la logica con cui le azioni vengono eseguite e dunque anche modificarne l'ordine o abortirle del tutto.

4.1.2 Gestione della sintesi vocale

La gestione della sintesi vocale coinvolge meno componenti essendo più semplice. Se non viene specificato, il Text-To-Speech viene gestito di default attraverso i Servizi Cognitivi Microsoft Azure, tuttavia c'è la possibilità di utilizzare un qualsiasi modello di deep learning allenato allo scopo. L'interazione con il modello o con Azure avviene in tutti i casi attraverso la classe **TtsManager**, singleton che contiene tutti i metodi necessari a:

- Specificare tutte le caratteristiche che l'audio deve avere
  - testo da sintetizzare
  - lingua
  - ECA che ha richiesto l'audio
- Generare la richiesta di conversione da testo a audio attraverso la creazione di un thread
- Gestire l'eventuale coda di messaggi

I messaggi che l'ECA può pronunciare possono essere definiti attraverso un XML o direttamente nel momento in cui viene generata la richiesta. La classe ha un riferimento a un oggetto di tipo **TtsModel** che è il vero responsabile della richiesta.

TtsModel

Anche in questo caso è fondamentale mantenere il codice il più isolato possibile in modo tale che in futuro sia semplice sostituire il tool utilizzato; per far questo viene utilizzata la classe astratta**TTSModel**, che fornisce un unico metodo **GenerateSpeechThread(SpeechInfo currentInfo)** nel quale le classi derivate, facendone l'override, definiscono il modo in cui avviene la richiesta audio.
Nella nostra implementazione la richiesta può essere di tipo diverso e se ne possono definire altre a piacere creando nuove classi che estendono TtsModel:

- Query verso un server esterno (es. Azure)

```
public override void GenerateSpeechThread(SpeechInfo currentInfo)
    {
        float[] audioData;

        // Creates an instance of a speech config with specified subscription
            key and service region.
        var config =
            SpeechConfig.FromSubscription(TtsManager.Instance.TtsServiceID,
            TtsManager.Instance.TtsZone);
```

```
        config.SpeechRecognitionLanguage = currentInfo.EcaLanguage;
        config.SpeechSynthesisVoiceName = currentInfo.EcaVoiceName;

        // Creates a speech synthesizer.
        using (var synthsizer = new SpeechSynthesizer(config, null))
        {
            // Starts speech synthesis, and returns after a single utterance is
                synthesized.
            var result =
                synthsizer.SpeakTextAsync(currentInfo.TextToSpeech).Result;

            // Checks result.
            if (result.Reason == ResultReason.SynthesizingAudioCompleted)
            {
                //synthesize audio data
            }
        }
    }
```

Listing 4.3: Implementazione di GenerateSpeechThread per Azure

- Richiesta verso un modello deep learning in locale (es. Tacotron-2):

```
    public override void GenerateSpeechThread(SpeechInfo currentInfo)
    {
        string text = currentInfo.TextToSpeech.Trim('"');
        string ecaName = currentInfo.EcaAnimator.Eca.Name;

        string line = "tts --text \"" + text + "\" --model_name
            tts_models/en/ljspeech/tacotron2-DDC " +
                "--out_path Assets\\Resources\\Audio\\" + ecaName + ".wav";

        //replace string request in file
        string[] fileLines = File.ReadAllLines(filename);
        using (StreamWriter writer = new StreamWriter(filename))
        {
            ...
        }

        //create and start a new process
        Process process = new Process();
        process.Start();
        process.WaitForExit();
    }
```

Listing 4.4: Implementazione di GenerateSpeechThread per Tacotron-2

Capitolo 5

Use Cases

In questo capitolo vengono illustrati gli use cases scelti per testare il sistema implementato. In ogni use case selezionato si è deciso di concentrarsi su un determinato aspetto del sistema, in modo da valutarlo nella maniera più accurata possibile e anche per capire quanto effettivamente i vari elementi che compongono il sistema nella sua interezza siano modulari. Un aspetto fondamentale che è stato attenzionato è stato quello di garantire la più alta immersione possibile da parte dell'utente, anche se questa è stata declinata in modi diversi nelle 3 applicazioni realizzate.

5.1 Simulazione Metro

Nella prima applicazione si è deciso di testare il sistema generando degli E^2CA autonomi e in grado di interagire con l'ambiente circostante senza bisogno di input da parte dell'utente. In questo modo è stato possibile valutare quanto il sistema sia flessibile anche a situazioni che non sono precisamente quelle per il quale è stato progettato, almeno per quanto riguarda la parte conversazionale.

Contesto

L'applicazione è ambientata in una stazione metropolitana, sulla banchina principale. Nell'ambiente sono presenti 2 tipi di oggetti con i quali gli agenti possono interagire:

- Distributori di bibite
- Biglietterie automatiche

All'inizio dell'applicazione gli E^2CA cominciano man mano ad arrivare sulla banchina. Ogni agente ha una logica interna per cui decide quale azione intraprendere in attesa dell'arrivo del treno; in particolare le azioni possibili sono 3:

- Andare al distributore di bibite per comprarne una
- Acquistare il biglietto del treno presso una biglietteria automatica
- Attendere il treno sulla banchina

Ognuna di queste azioni ha una sua probabilità e possono essere fatte anche tutte in sequenza. L'obiettivo finale di tutti gli agenti rimane comunque prendere il treno non appena arriva e trovare un posto libero al suo interno dove sedersi o aggrapparsi. Nel momento in cui il treno raggiunge la banchina viene generato un evento che gli E^{2}CA ricevono; a quel punto gli scenari possibili sono principalmente 2:

- Se l'agente sta eseguendo un'azione abortibile, allora smette di farla e si dirige direttamente verso il treno
- Se invece l'azione non è abortibile, allora finisce l'azione attuale e successivamente si dirige verso il treno, eliminando dalla coda eventuali altre azioni (sempre a patto che siano abortibili)

Implementazione

Per questo use case si è reso molto utile l'utilizzo delle **ECACompositeAction**; tutte le azioni che possono eseguire gli E^{2}CA hanno una logica interna piuttosto complessa che hanno portato alla loro suddivisione in microazioni per una gestione più semplice.
Per spiegare meglio il concetto qui di seguito viene riportata l'implementazione dell'ECACompositeAction **BuyTicket**.

```
public class BuyTicket : ECACompositeAction
{
   //constructor
   public BuyTicket(Passenger eca) { ... }

   protected override void CreateActionList()
   {
      GoToVendingMachine();
      CompleteQueueing();
      SelectTicket();
   }

   private void GoToVendingMachine(){
      GoToStage s = new GoToStage(queuePosition);
      ECAAction action = new ECAAction(eca, s);
      actions.Add(action);
   }
```

```
    private void CompleteQueueing()
    {
        ManageQueue m = new ManageQueue();
        actions.Add(m);
    }

    private void SelectTicket()
    {
      SelectTicket s = new SelectTicket();
      actions.Add(s);
    }
}
```

Listing 5.1: Implementazione BuyTicket.cs

Questa è composta da 3 ECAAction:

- *GoToVendingMachine:* è l'azione iniziale che permette all'agente di raggiungere la biglietteria designata
- *ManageQueue:* permette all'agente di gestire l'avanzamento nell'eventuale fila; ogni volta che un E^{2}CA prende il biglietto ed esce dalla fila chiunque altro sia in questa fase dell'azione avanzerà e se sarà il primo nella fila comincerà la prossima azione
- *SelectTicket*: azione finale in cui l'E^{2}CA seleziona e compra il biglietto e infine esce dalla fila

Nell'esempio presentato sopra è possibile notare come nel caso dell'azione creata all'interno di *GoToVendingMachine()* questa venga generata direttamente all'interno del metodo in quanto molto semplice e composta da un unico stage. Nel caso delle altre 2 azioni invece la logica interna è più complessa e quindi sono stati creati degli script appositi a cui viene delegato il setup dell'azione e la gestione interna. Di seguito è riportata la pseudo-implementazione delle azioni *SelectTicket()* e *ManageQueue():*

```
public class SelectTicket : ECAAction
{
    public SelectTicket(ECA eca, VendingMachine machine) : base(eca)
    {
      //this action is mandatory
      //for taking the train
        canAbort = false;
    }

    public override void SetupAction()
    {
      List<ECAActionStage> stages = new List<ECAActionStge>();
```

```
        PressStage useScreen = new PressStage(machine.screen);
        PressStage pressButton = new PressStage(machine.button);
        PickStage takeTicket = new PickStage(machine.ticket);
        DropStage dropTicket = new DropStage(eca.pocket);
        GoToStage exitQueue = new GoToStage(machine.exitPoint);

        stages.Add(useScreen);
        stages.Add(pressButton);
        stages.Add(takeTicket);
        stages.Add(dropTicket);
        stages.Add(exitQueue);

        SetStages(stages);
    }
}
```

Listing 5.2: Implementazione SelectTicket.cs

```
public class ManageQueue : ECAAction
{
    public ManageQueue(Passenger eca, QueueableObject qo)
    :base(eca)
    {
      ...
    }

    public override void StartAction()
    {
        inQueue = true;

        //Se e' il primo della fila
        if(eca.ecaTurn == 0)
        {
         //Avanza fino alla prima posizione
        }
      else
      {
         //Si iscrive all'evento che segnala
         //la possibilita' di un
         //avanzamento nella fila
         queueableObject.GoAhead += OnGoAhead;
      }
    }

    //Chiamato quando un ECA davanti
    //abbandona la coda
    protected void OnGoAhead(object sender, EventArgs e)
    {
```

```
        //Avanzo nella fila
        eca.ecaTurn--;
        GoToStage s = new GoToStage(qo.nextPosition);
        ECAAction advance = new ECAAction(eca, s);
        advance.StartAction();

      //Se l'ECA diventa della fila
       if(eca.ecaTurn == 0)
       {
           advance.CompletedAction += OnArrivedToMachine;
           queueableObject.GoAhead -= OnGoAhead;
       }
    }

    //Chiamato quando l'ECA raggiunge
    //il primo posto della fila
    protected void OnArrivedToMachine(object sender, EventArgs e)
    {
      //Termina l'azione
        EndAction();
    }
}
```

Listing 5.3: Implementazione ManageQueue.cs

La possibilità di delegare la complessità nell'implementazione dell'azione permette di mantenere il codice della classe ECA più semplice e anche di definire delle condizioni di transizione tra un'azione e l'altra facilmente.

5.2 Simulazione ACLS protocol

Nel secondo use case sviluppato si è deciso di testare il sistema nell'implementazione di un'applicazione di realtà virtuale nel campo del training medico. Gli E^2CA in questo caso hanno un ruolo attivo e reagiscono agli input vocali dell'utente.

Contesto

L'applicazione permette a un medico di apprendere tutti i vari step necessari nel caso in cui si trovi nella situazione in cui è necessaria una rianimazione. L'utente è il team leader e dovrà impartire ordini agli altri membri del team utilizzando la propria voce. Ogni E^2CA ha un ruolo ben preciso e risponderà con l'azione corrispondente all'intent riconosciuto.

Protocollo ACLS

L'acronimo ACLS sta per *Advanced Cardiovascular Life Support* (supporto avanzato di rianimazione cardiovascolare), cioè il protocollo medico-sanitario che viene utilizzato in presenza di un paziente che si trovi in stato di arresto cardiaco e complicazioni cardiovascolari. La procedura si può riassumere sostanzialmente in due parti:

- **Intervento primario:** gli operatori formano un quadro generale dello stato del paziente e, dopo aver compreso le cause dell'arresto cardiaco, delineano una possibile terapia da eseguire
- **Intervento secondario:** una volta stabilizzate le condizioni si continua con la terapia farmacologica e con altri eventuali interventi

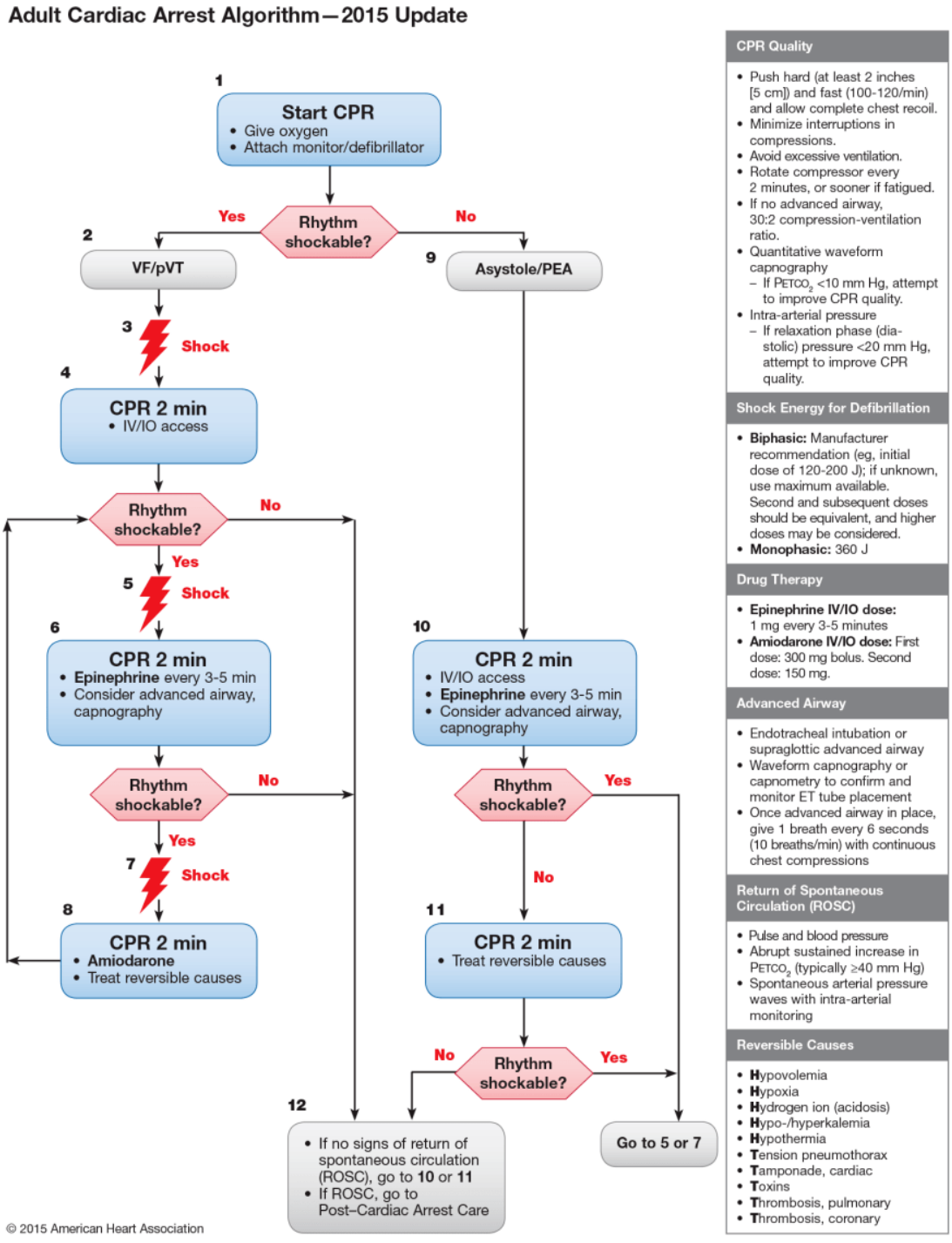


Figura 5.1: Protocollo ACLS

Nell'applicazione sviluppata vengono eseguiti solo gli step presenti nello schema in figura.
Il protocollo prevede che siano presenti più persone, ognuna delle quali ha un ruolo ben preciso da svolgere. I vari ruoli sono elencati qui di seguito:

- Team leader: nell'applicazione è l'utente, prende tutte le decisioni relative al trattamento ed eventualmente si assume la responsabilità dei ruoli non assegnati

- Addetto alle medicine: si occupa di iniettare le medicine necessarie a salvare il paziente

- Addetto alle vie aeree: esegue la ventilazione

- Addetto al defibrillatore: controlla i parametri vitali del paziente attaccando gli elettrodi al monitor ed è anche il responsabile del defibrillatore, nel caso in cui sia necessario il suo utilizzo

- Addetto al rilevamento dei tempi: registra i tempi dell'intervento e della somministrazione dei farmaci, registra la frequenza e la durata delle interruzione delle compressioni e comunica i tempi al team leader e a tutto il team

- Addetto alle compressioni: valuta il paziente ed esegue dei cicli di compressioni toraciche per 2 minuti

Implementazione

Per questo use case l'interazione tra utente e sistema avviene unicamente attraverso l'utilizzo della voce. L'utilizzatore infatti può impartire ordini agli E^2CA semplicemente parlando; una volta riconosciuto il testo questo viene analizzato per capire a cosa si riferisce e successivamente al riconoscimento l'agente virtuale interessato eseguirà l'azione indicata a voce. Ogni E^2CA infatti è iscritto agli intent relativi al suo ruolo; questo significa che soltanto il giusto agente reagisce a un determinato ordine impartito dall'utente. Ogni qual volta viene eseguita un'azione sbagliata, cioè ogni volta che non si segue il protocollo nel giusto ordine, lo stato del paziente peggiora e saranno necessarie più ripetizioni del protocollo per riuscire a salvarlo.
Per testare ulteriormente il sistema e in particolare l'interazione tra i vari agenti conversazionali è stato inserito un ulteriore E^2CA nella scena, con il compito di rifornire le medicine che mancano. Dunque ogni volta che l'operatore addetto alle iniezioni non ha più medicine a disposizione lo comunica a questo agente che a sua volta andrà a prendere la medicina mancante e la riposiziona al suo posto.

5.3 Simulazione paziente

Nel terzo use case si è testata l'efficacia del sistema nel gestire una conversazione con l'utente e la possibilità di richiedere delle azioni all'E^{2}CA senza dover necessariamente programmare ma direttamente da un file XML.

Contesto

L'applicazione è ambientata nella stanza di una clinica in cui si trova un paziente E^{2}CA. L'utente impersona un dottore e può conversare con il paziente su certi argomenti, corrispondenti ad altrettanti intent:

- Presentarsi come dottore
- Chiedere di sedersi
- Chiedere come sta
- Chiedere come stanno i bambini del paziente
- Chiedere informazioni sul meteo
- Annunciare una brutta notizia
- Spiegare le complicazioni del cancro
- Annunciare una nuova cura

La tipologia di intent riconosciuto influeza sia l'emozione dell'agente che di conseguenza le sue risposte. Lo stato d'animo del paziente è osservabile attraverso una ruota presente all'interno della scena

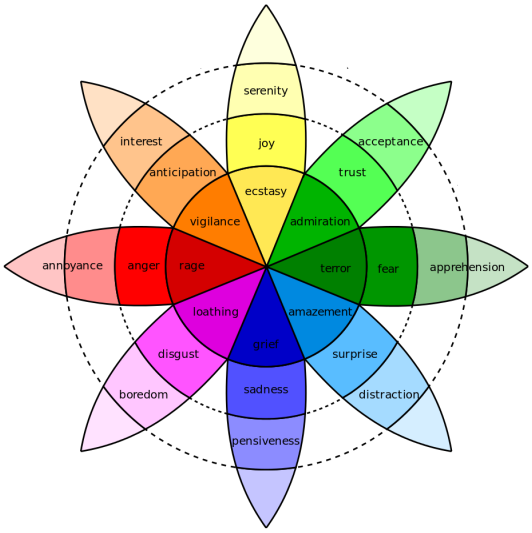


Figura 5.2: Ruota di Plutchik

Implementazione

Come nel secondo use case, anche in questo caso l'interazione tra utente e E²CA avviene unicamente attraverso l'utilizzo della voce. Ogni intent riconosciuto dal paziente influenza il suo stato d'animo in modo diverso attraverso le **Appraisal-Variables** definite in un file XML.

```
<model>
  <Good>
    <Joy>+0.1</Joy>
    <Sadness>-0.1</Sadness>
    <Surprise>-0.1</Surprise>
    <Anger>-0.1</Anger>
  </Good>
 </model>
```

Listing 5.4: Definizione della AppraisalVariable "Good"

```
public override void Handle(Intent intent)
{
  //Prima aggiorna lo stato d'animo
    switch (intent.IntentName)
    {
        case BAD_NEW:
            EmotionManager.UpdateEmotion(AppraisalVariables.UnexpectedNegative);
            break;
        case CHILDREN:
            EmotionManager.UpdateEmotion(AppraisalVariables.Nice);
            break;
        case EXPLAIN_DISEASE:
            EmotionManager.UpdateEmotion(AppraisalVariables.Bad);
            break;
        case PRESENTATION:
            EmotionManager.UpdateEmotion(AppraisalVariables.Nice);
            break;
        case CURE:
            EmotionManager.UpdateEmotion(AppraisalVariables.Good);
            break;
        case GENERAL_HEALTH:
            break;
        case SIT:
            HandleSit();
            break;
        case NONE:
            break;
    }

    //Poi riproduce il messaggio
    SendMessage(intent.IntentName);
}
```

Listing 5.5: Implementazione del metodo Handle() all'interno della classe ConversationalPatient.cs

I vari messaggi che l'agente può riprodurre sono definiti attraverso un file XML attraverso l'uso di alcuni attributi:

- **emotion:** permette di definire l'emozione associata a quel messaggio. A seconda dello stato d'animo dell'E^{2}CA verrà preso il messaggio con la relativa emozione, in assenza di questo verrà preso invece il messaggio che non ha nessuna emozione specificata
- **action:** permette la definizione di azioni che l'E^{2}CA deve compiere insieme alla riproduzione del messaggio. Le azioni possibili sono:
  - Guardare un oggetto o in una direzione ("LookAt")
  - Puntare un oggetto con la mano ("PointAt")
  - Prendere un oggetto in mano ("PickUp")
  - Raggiungere un punto della scena ("GoTo")
  - Sedersi ("Sit")

```
<Weather emotion="Joy" action="LookAt@Window#0.5,PointAt@Sun#0.5">Oh I love
    this weather!</Weather>
<Weather emotion="Anger">It is not time to talk about weather doc!</Weather>
<Weather emotion="Disgust" action="PointAt@MainCamera">I hate the sun and I
    hate you too.</Weather>
<Weather emotion="Fear" action="LookAt@SadEmpty">I don't see the light in
    the tunnel.</Weather>
<Weather emotion="Sadness" action="LookAt@Window">I prefer the rainy
    days.</Weather>
<Weather action="LookAt@Window#0.5,PointAt@Sun#0.5">I love the sun, look at
    that!</Weather>
```

Listing 5.6: Definizione di messaggi da XML

Gestione azioni da XML

Le azioni richiamabili da XML tramite l'attributo "action" devono essere definite dal programmatore a priori e devono utilizzare tutte lo stesso tipo di simboli e separatori. In particolare:

- Il simbolo **"@"** indica di norma l'oggetto con il quale l'E^{2}CA deve interagire; ad esempio nel caso dell'azione PickUp la parola che segue il simbolo @ indica il tag dell'oggetto da raccogliere
- Il simbolo **"#"** seguito da un numero decimale compreso tra 0 e 1 indica che quell'azione ha una probabilità di essere eseguita pari a quel numero (in %, ad esempio se il numero in question è 0.5 allora quell'azione ha il 50% di

probabilità di essere eseguita). Questo permette di definire delle azioni opzionali che non vengono sempre eseguite e che quindi rendono il comportamento dell'agente meno prevedibile

- Il simbolo **","** permette di definire più azioni, per un unico messaggio che possono essere opzionali o sequenziali:
  - Se le azioni definite sono tutte seguite dal simbolo "#" allora saranno tutte opzionali e ne verrà eseguita una sola di queste
  - Se le azioni definite non sono seguite dal simbolo "#" allora verranno eseguite tutte e saranno inserite nella ECAActionList sequenzialmente
  - Se alcune azioni definite sono seguite da "#" e altre no allora quelle senza probabilità verranno messe in coda e tra le altre ne sarà eseguita opzionalmente al massimo una e verrà messa in coda

La possibilità di definire le azioni che E^2CA può o deve eseguire attraverso un semplice file XML è molto importante in quanto è un primo passo verso un approccio che non prevede alcun tipo di programmazione per poter creare un'applicazione simile a questa.

Capitolo 6

Test

Sono state effettuati una serie di Test sugli use cases descritti precedentemente. Per quanto riguarda la parte della Metro i test sono stati solo ed esclusivamente sulle prestazioni dell'applicazione mentre per i successivi due i test sono stati effettuati su persone che, volontariamente, hanno dato il loro consenso a testare l'applicazione.

6.1 Metro

6.1.1 Protocollo sperimentale

Di questo use case sono stati effettuati solo test oggettivi riguardanti le prestazioni dell'applicazione con analisi basata su Stress Test. Sono stati utilizzati i dati del profiler di Unity in varie condizioni che testassero la qualità del sistema a reggere diverse situazioni.

Materiale

Il computer utilizzato per lo use case della Metro è così assemblato:

- **Processore**: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
- **RAM**: 32GB
- **Sistema Operativo**: Windows 64bit, processore basato x64
- **Scheda Grafica**: NVIDIA GeForce RTX 2070 SUPER 16GB

Si tratta di un computer molto performante, ma non il top di gamma per quanto riguarda la scheda grafica.
Come pipeline di rendering è stata utilizzata la **High Definition Render Pipeline** (HDRP) che risulta molto dispendiosa a livello di prestazioni.

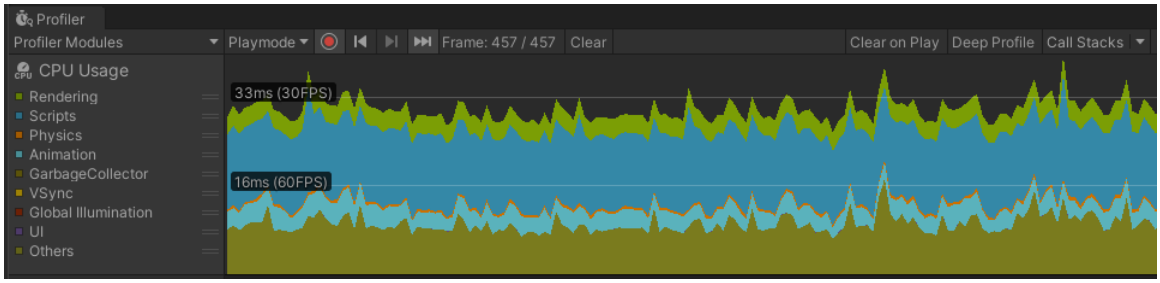
Design

I test sono stati effettuati creando situazioni differenti: il numero di ECA è stato cambiato per capire quale fosse il numero adatto per evitare dei cali di frame rate e per testare i comportamenti dei vari agenti virtuali nell'ambiente e fra di loro. I test quindi vanno a testare sia le prestazioni che la capacità di adattamento del sistema a situazioni limite.

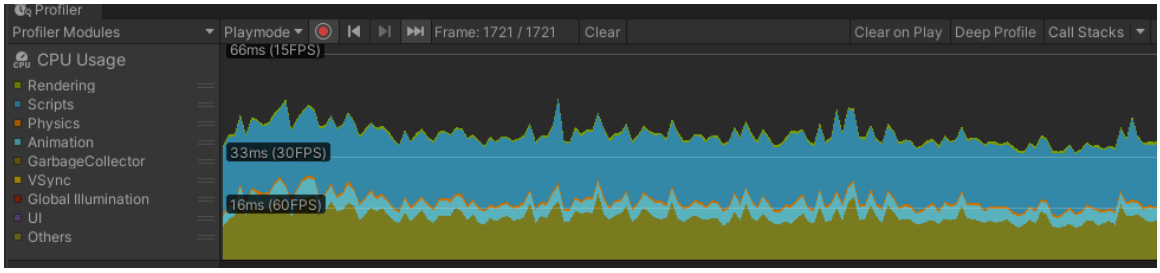
Dati oggettivi

Per questo use case sono stati analizzati solo dei dati oggettivi. In particolare, tramite il profiler di Unity abbiamo analizzato le prestazioni in tre situazioni differenti. Bisogna considerare che il funzionamento dell'applicazione era stato pensato per circa 30 E^2CA.
Nelle seguenti immagini si vedranno le prestazioni in occasioni di prove diverse, in particolare sono state prese in considerazione due variabili: il numero degli E^2CA e render della camera acceso o spento.

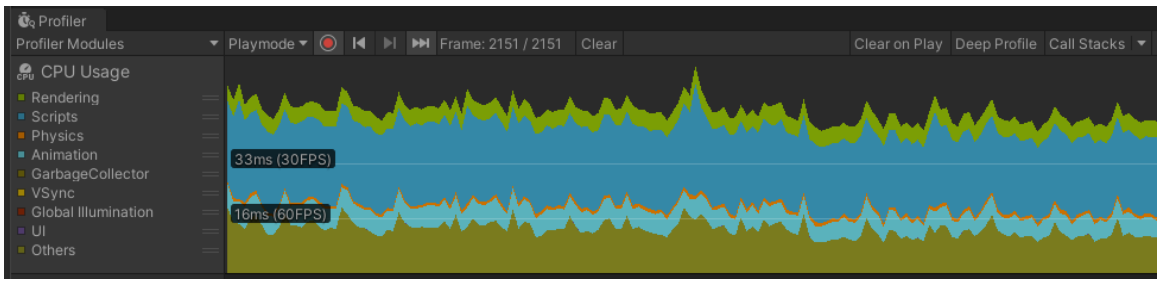
Numero di E^2CA: 80 Camera: Off



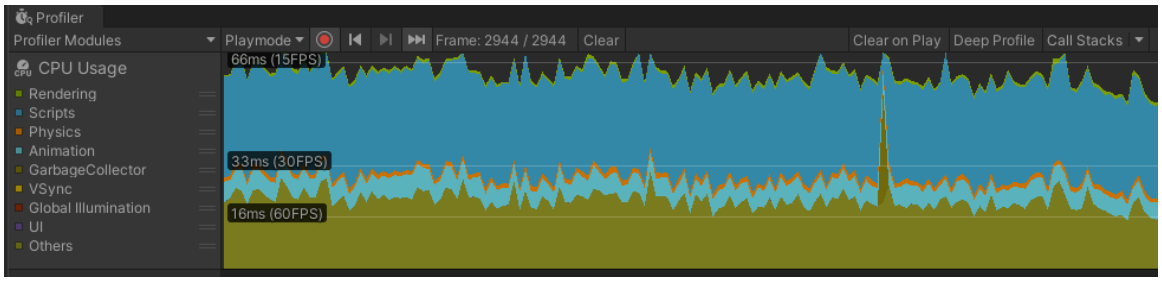
Numero di E^2CA: 80 Camera: Off



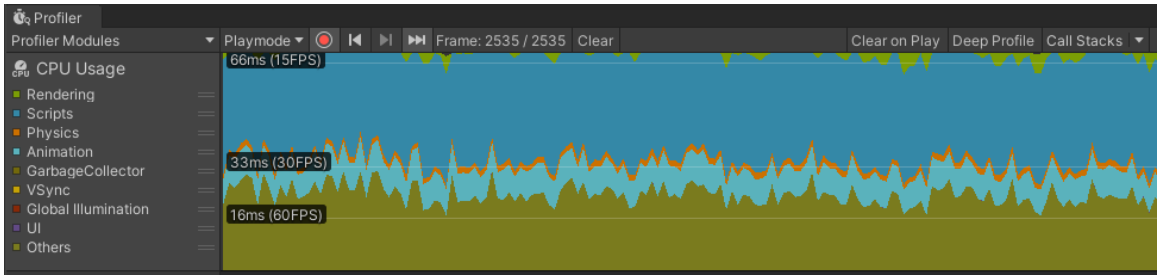
Numero di E^{2}CA: 130 Camera: Off



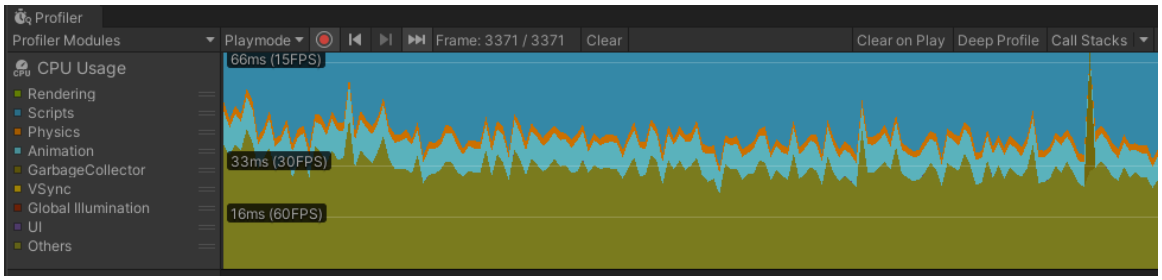
Numero di E^{2}CA: 130 Camera: On



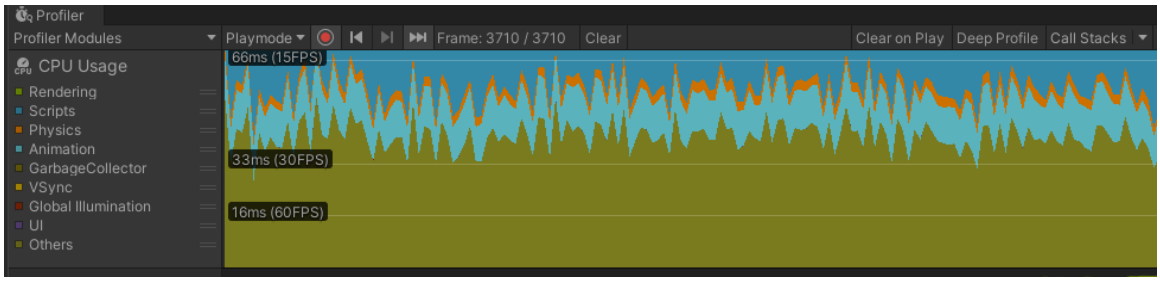
Numero di E^{2}CA: 180 Camera: Off



Numero di E^{2}CA: 180 Camera: On



Numero di E^{2}CA: 230 Camera: Off



Numero di E^{2}CA: 230 Camera: On

Figura 6.8: Screenshots del profiler durante i test

Si può vedere dalle immagine che le prestazioni risultano buone fino a circa 130 E^{2}CA dove si ha una media di circa 22 FPS di rendering. Considerando le prestazioni a camera spenta anche la parte a 180 può essere presa in considerazione come decente poiché si ha una media di circa 18 FPS.
Riguardo la prestazione delle animazioni vediamo che si ritrova una notevole differenza fra camera spenta e accesa con 230 di numero, questo poiché probabilmente la renderizzazione di animazioni e contatti così vicini, in caso di notevole numero

da gestire, risulta poco accurata da parte del sistema.
Utilizzando pipeline di render meno dispendiose dell'HDRP sicuramente si raggiungerebbero dei cali di frame rate minori, ma abbiamo voluto effettuare un vero e proprio stress test dell'applicazione.

6.2 ACLS

6.2.1 Protocollo Sperimentale

Partecipanti

Il protocollo di test è stato effettuato su xx partecipanti. Il gruppo era molto eterogeneo così in modo da avere qualitativamente un risultato consono per la ricerca. Fra questi partecipanti solo una era un medico e quindi a conoscenza della procedura dell'ACLS e aveva già avuto esperienza nel parlare con un paziente malato.

Ambiente e Materiale

Tutti gli esperimenti sono stati effettuati in una stanza molto silenziosa e senza fare in modo che i partecipanti ascoltassero i comportamenti di quelli precedenti.
Il computer utilizzato per i test di ACLS e parte Conversazionale non è eccessivamente performante ma comunque ha retto in maniera adeguata le varie applicazioni, si tratta di un pc così assemblato:

- **Processore**: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
- **RAM**: 16GB
- **Sistema Operativo**: Windows 64bit, processore basato x64
- **Scheda Grafica**: NVIDIA GeForce GTX 960M 4GB

Design

Prima di far effettuare l'esperienza all'utente abbiamo spiegato come doversi comportare e cosa fare, in particolare dare istruzioni vocali al sistema e controllare che i comportamenti degli E^2CA. In questo use case l'obiettivo era testare il sistema nel suo complesso, ovvero misurare qualitativamente le animazioni, le risposte e in generale l'interazione con gli E^2CA. Per effettuare il test è stato dato all'utente il seguente set di istruzioni da impartire al sistema:

- "Attacchiamo il monitor"

- "Diamo ossigeno"
- "Effettua un massaggio cardiaco"
- "Controlla lo stato del paziente"
- "Diamo una scarica"
- "Effettua un altro massaggio cardiaco"
- "Effettuiamo l'accesso venoso"
- "Battito?"
- "Scarica"
- "Massaggio cardiaco"
- "Somministriamo 1mg di epinefrina"
- "Intubiamo il paziente"
- "Stato del paziente"
- "Iniettiamo dell'amiodarone"
- "Battito?"

Per impartire un'istruzione l'utente doveva premere una volta il tasto **R** e poi parlare.

Dati oggettivi

Durante la fase di test è stato creato a runtime e in maniera diversa per ogni tester un file di Log che prendesse in considerazione la frase pronunciata dal tester e l'intent riconosciuto. In questo modo abbiamo testato quanto le frasi inserite come intent fossero precise e come tutto il sistema di riconoscimento degli intent fosse flessibile.

6.2.2 Questionari

Dopo la conclusione dell'esperienza abbiamo fatto compilare dei questionari in modo tale da raccogliere dei dati soggettivi su di essa. Per effettuare il questionario dell'esperienza di ACLS abbiamo fatto riferimento al tipo di questionario **SASSI**, cioè Subjective Assessment of Speech System Interfaces. I dati soggettivi sono stati raccolti per capire quanto le interazioni con l'E^2CA fossero efficienti e per testare

in generale il sistema. Dei futuri test, dopo che verrà effettuata un lavoro di interfaccia grafica, potrebbero essere sul testare in quanto tempo e quanto facilmente si possa creare uno use case. Attualmente quel tipo di test non era sperimentabile con gente esterna, visto il fatto che per costruire uno use case del genere servono diverse competenze di programmazione.

SASSI

Il questionario SASSI è un tipo di questionario molto diffuso per testare sistemi interattivi e anche interfacce conversazionali. La composizione è di 34 domande divise per macrotematiche:

- System Response Accuracy: domande che vanno a testare quanto il sistema sia *responsive*. (Es. L'interazione con il sistema è consistente?).
- Likeability: domande che prendono in considerazione quanto l'applicazione e l'interazione con essa sia gradevole. (Es. Ti è piaciuto utilizzare il sistema?).
- Cognitive Demand: domande che considerano la risposta cognitiva dell'utente con il sistema. (Es. Serve un alto livello di concentrazione per utilizzare il sistema?)
- Annoyance: domande che vanno a testare quanto il sistema sia noioso. (Es. L'interazione con il sistema era irritante?).
- Habitability: domande indirizzate per capire quanto si ci sentisse confidenti nell'utilizzare il sistema.
- Speed: domande che vanno a testare quanto il sistema sia veloce.

6.2.3 Risultati

Log Intent

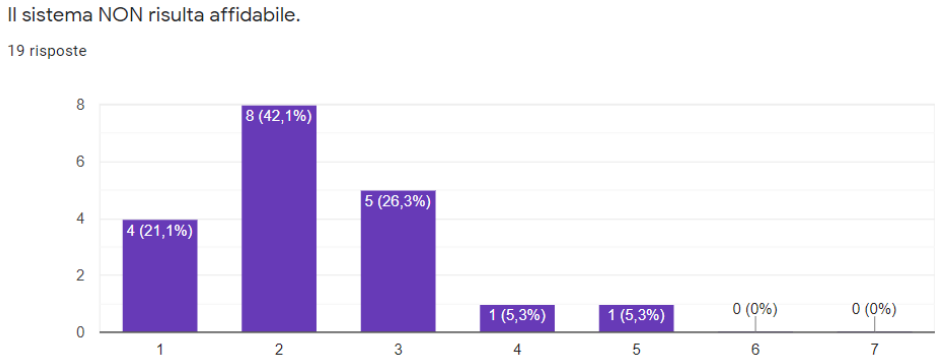
La precisione del sistema risulta molto alta, mentre gli score (che definiscono quanto il sistema sia sicuro di riconoscere un determinato intent) hanno una media di circa 0.5, il che vuol dire che nel riconoscere gli intent la sicurezza del sistema è di circa la metà del totale. Bisogna specificare, però, che le frasi che abbiamo inserito nell'esperienza non sono tutte presenti all'interno degli intent di LUIS, in modo che si potesse testare anche la flessibilità del sistema. Vista l'alta precisione del riconoscimento siamo molto soddisfatti dei risultati.
Di seguito viene inserito il risultato del Log per le varie esperienze: il primo numero definisce la percentuale di precisione su quanti intent sono stati riconosciuti sul

totale delle frasi dette, mentre la seconda parte definisce la media degli score dei vari intent.

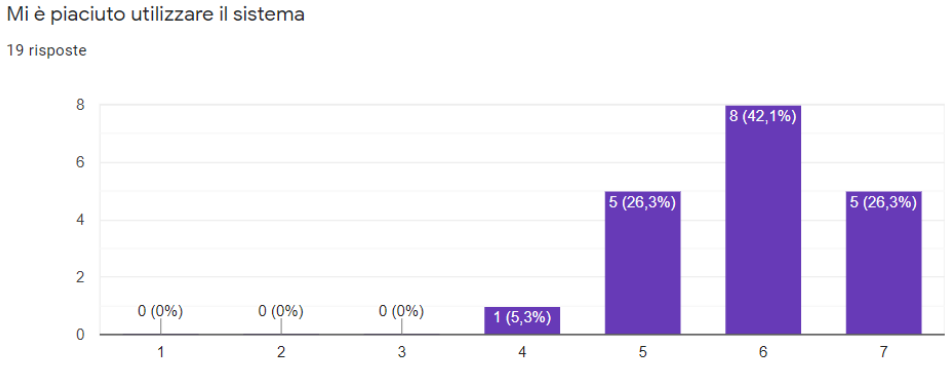
Nr. Test	Precisione	Media Score
1	86%	0.52
2	100%	0.56
3	100%	0.46
4	100%	0.46
5	100%	0.56
6	100%	0.47
7	100%	0.50
8	100%	0.56
9	93%	0.52
10	93%	0.48
11	100%	0.50
12	100%	0.58
13	93%	0.56
14	100%	0.54
15	93%	0.57
16	100%	0.56
17	100%	0.54
18	93%	0.57
19	94%	0.55

Risultati Questionari

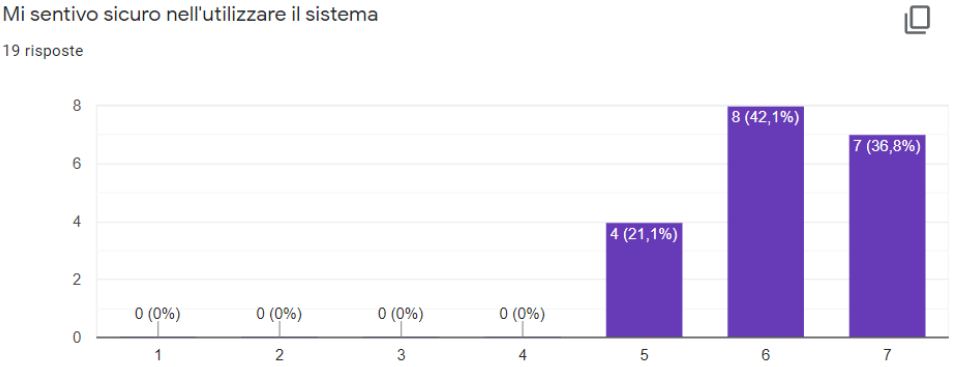
Di seguito vengono riportati i risultati principali del questionario.
Abbiamo selezionato una domanda principale e significativa per ogni macrotematica.



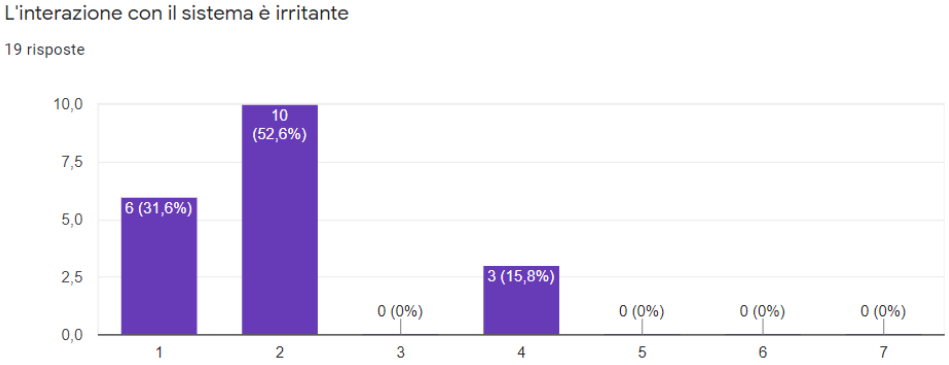
System Response Accuracy: domanda sull'affidabilità, 1 indica "Estremamente in disaccordo", 7 invece "Estremamente d'accordo"



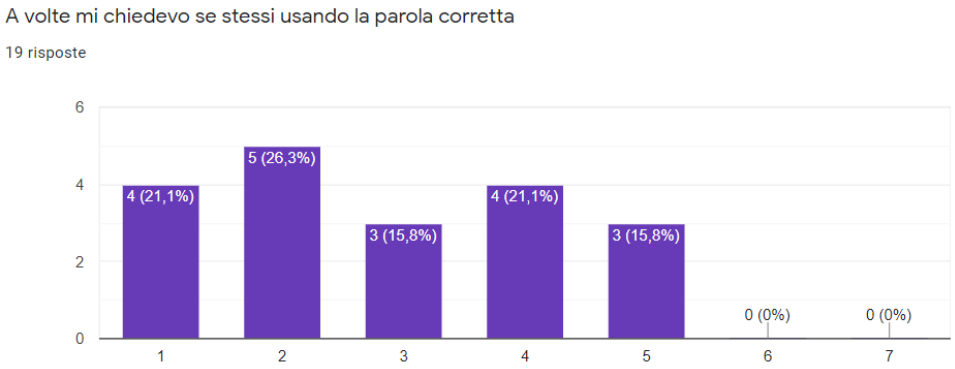
Likeability: domanda sulla piacevolezza del sistema, 1 indica "Estremamente in disaccordo", 7 invece "Estremamente d'accordo"



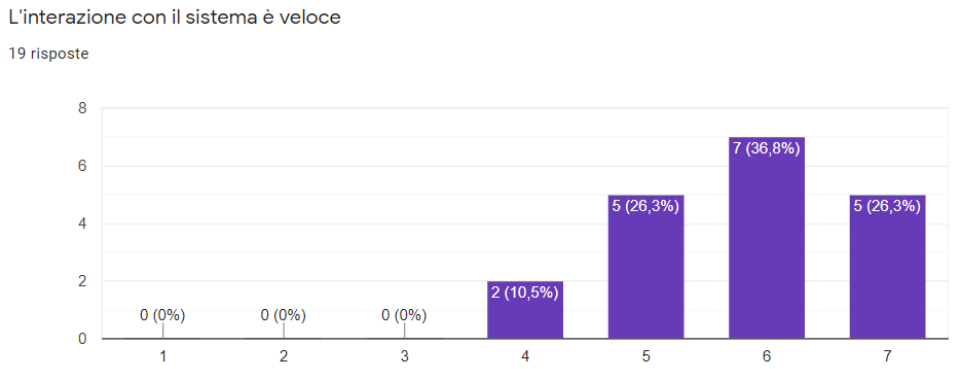
Cognitive Demand: domanda sulla sicurezza nell'utilizzare il sistema, 1 indica "Estremamente in disaccordo", 7 invece "Estremamente d'accordo"



Annoyance: domanda sull'irritabilità del sistema, 1 indica "Estremamente in disaccordo", 7 invece "Estremamente d'accordo"



Habitability: domanda sulla sicurezza nella conversazione, 1 indica "Estremamente in disaccordo", 7 invece "Estremamente d'accordo"



Speed: domanda sulla velocità del sistema, 1 indica "Estremamente in disaccordo", 7 invece "Estremamente d'accordo"

Figura 6.14: Grafici derivanti dai test ACLS

I risultati dei questionari sono positivi in quasi tutte nei vari macrotemi affrontati, la parte con risultati negativi è quella dell'habitability, probabilmente il sistema va migliorato in alcune parti per fare in modo che venga reso più confident per l'utente.

6.3 Parte Conversazionale

Per le sottosezioni di "Partecipanti" e "Ambiente e materiali" si possono guardare quelle scritte il precedenza per l'ACLS perché il test è stato sottoposto alle stesse persone, nello stesso ambiente e con lo stesso computer.

6.3.1 Design

In questo use case l'obiettivo è testare principalmente la naturalezza della voce dell'E^{2}CA e l'elasticità nel riconoscimento del corretto intent da parte dell'E^{2}CA. Nell'esperienza il tester ha impersonificato un dottore e gli sono stati forniti degli argomenti per poter dialogare con un paziente E^{2}CA. Per poter fare un confronto tra le 2 principali soluzioni per la sintesi vocale che sono state individuate, il tester ha eseguito l'esperienza 2 volte, rispettivamente ognuna con un TTS (Text-To-Speech) diverso, e ha compilato un questionario per ognuna, in modo da poter valutare le differenze fra le due soluzioni.
Il primo tipo di TTS sfrutta il servizio "Voce" di Azure mentre il secondo è una rete neurale allenata con Tacotron 2.
Qui di seguito gli argomenti e le frasi di esempio date al partecipante:

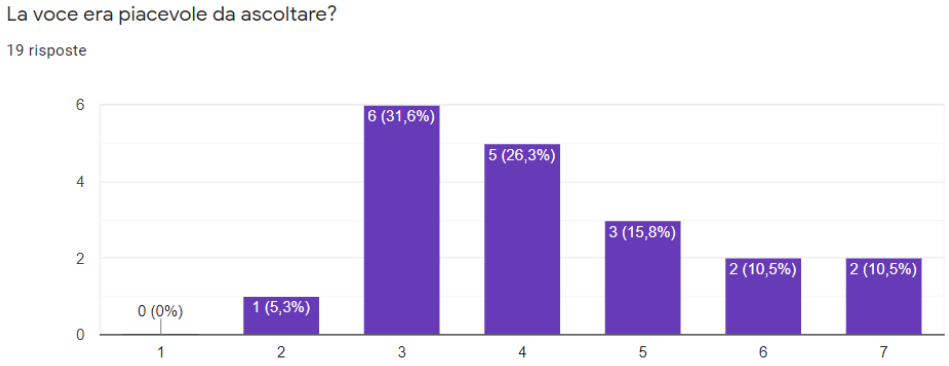
- Presentarsi come dottore (es. "Hello, I am the doctor")
- Chiedere di sedersi (es. "Why don't you sit?")
- Chiedere come stanno i bambini del paziente. (es. "How are your children?")
- Chiedere informazioni sul meteo (es. "What do you think about the weather?")
- Chiedere come sta (es. "how are you today?")
- Annunciare una brutta notizia (es. "I have bad news for you")
- Spiegare le complicazioni del cancro (es. "Your cancer is growing")
- Annunciare una nuova cura (es. "Don't be afraid, we can try a new cure")

6.3.2 Questionari

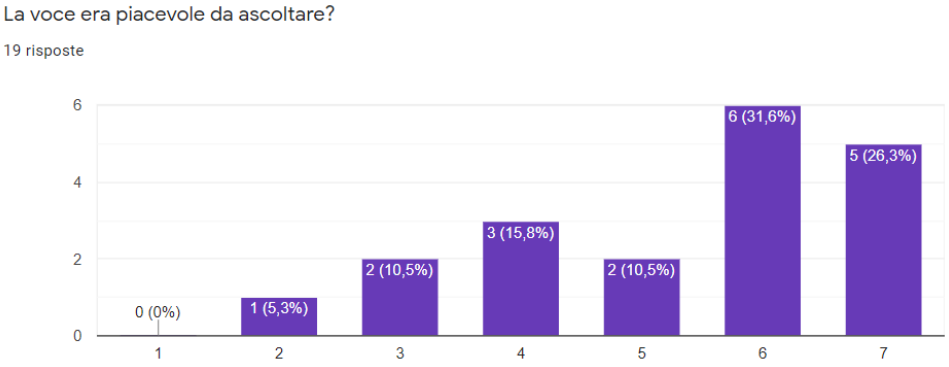
La modalità di sottomissione ai questionari è stata diversa rispetto allo use case precedente. In questo caso vengono utilizzati due TTS diversi ed il fine dei test era quello di capire quale dei due TTS fosse migliore in termini di comprensibilità del parlato ed in termini di naturalezza.
Per arrivare a ciò abbiamo fatto provare, in maniera casuale, prima l'esperienza con un TTS e poi con l'altro, facendo rispondere ai questionari una volta finita un'esperienza (non a fine totale, ma alla fine di un'esperienza con un TTS).
Il tipo di questionario che abbiamo utilizzato è il Mean Opinion Scale-Expanded (MOS-X), molto utilizzato per valutare voci sintetiche.
Il questionario è formato da 15 domande riguardanti la voce. Alla fine è stata aggiunta una domanda per capire quanto la conversazione fosse naturale, visto che la velocità di risposta dei due TTS è diversa.

6.3.3 Risultati

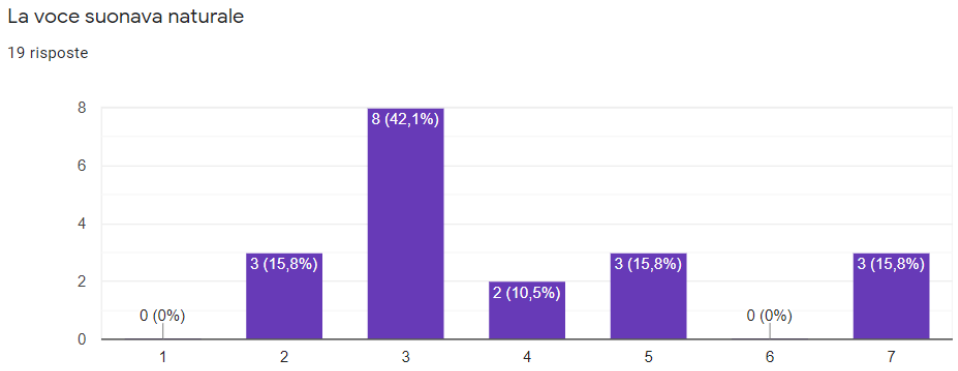
In questa sottosezione andremo a far vedere le differenze di percezione soggettiva che si sono riscontrate in maniera maggiore all'interno del questionario. Non saranno mostrate le risposte riguardanti la comprensione del parlato perché in entrambi i casi i risultati sono stati positivi.



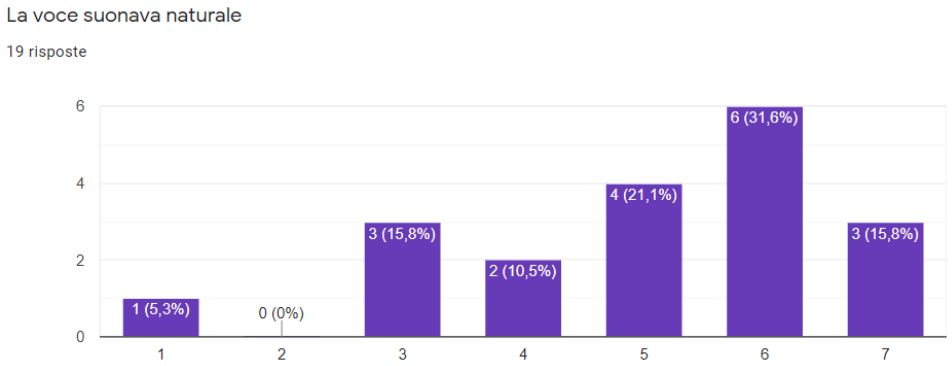
Piacevolezza Azure, 1 indica "Per niente piacevole", 7 invece "Molto piacevole"



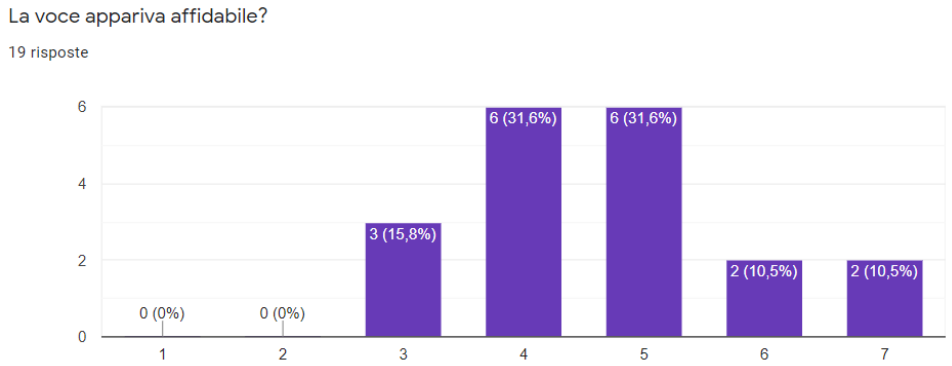
Piacevolezza Tacotron, 1 indica "Per niente piacevole", 7 invece "Molto piacevole"



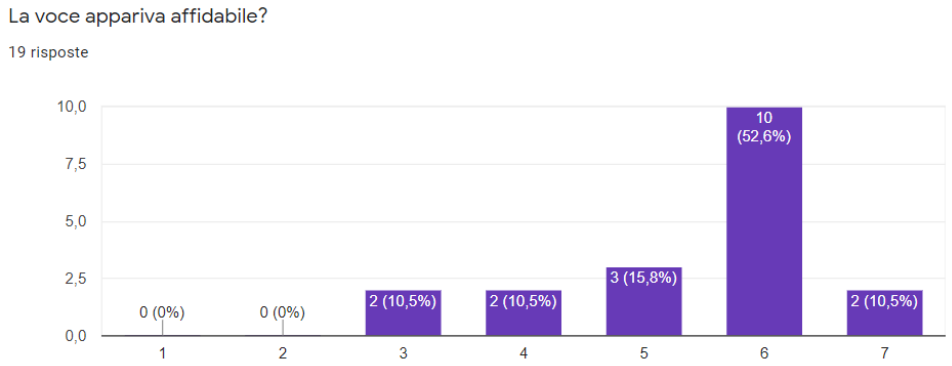
Naturalezza Azure, 1 indica "Molto innaturale", 7 invece "Molto naturale"



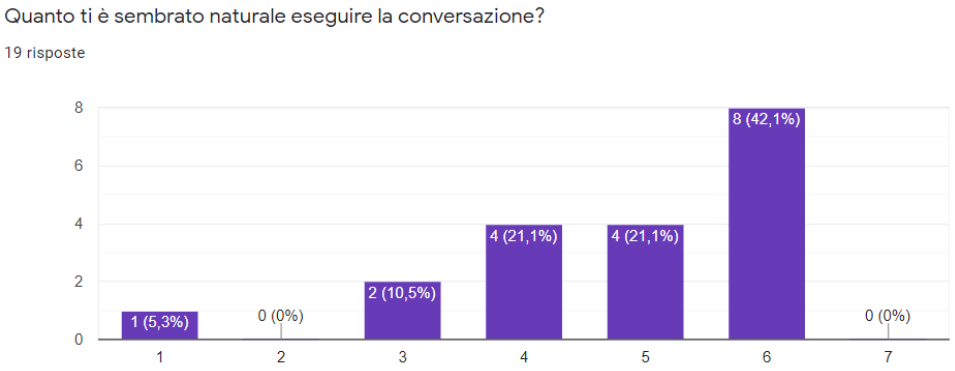
Naturalezza Tacotron, 1 indica "Molto innaturale", 7 invece "Molto naturale"



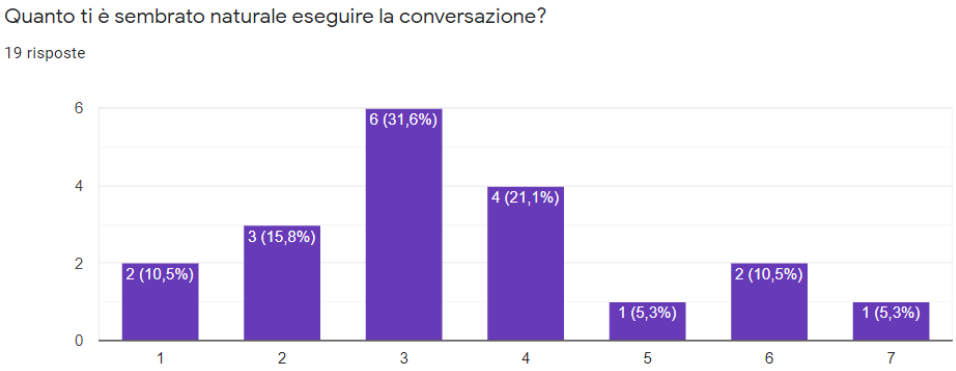
Affidabilità Azure, 1 indica "Per niente affidabile", 7 invece "Molto affidabile"



Affidabilità Tacotron, 1 indica "Per niente affidabile", 7 invece "Molto affidabile"



Naturalezza conversazione Azure, 1 indica "Per niente naturale", 7 invece "Molto naturale"



Naturalezza conversazione Tacotron, 1 indica "Per niente naturale", 7 invece "Molto naturale"

Figura 6.22: Grafici derivanti dai test su TTS

Visti i risultati Tacotron 2 risulta una voce molto più umana e definita rispetto a quella di Azure, ma il ritardo nella risposta di Tacotron 2 rende la conversazione molto meno naturale rispetto ad Azure.

Capitolo 7

Conclusioni

Il lavoro di Tesi è stato pensato fin dall'inizio per essere il più modulabile possibile. Dopo aver creato le basi del framework tali da poter creare in maniera facile e veloce delle azioni e la loro logica da seguire, abbiamo effettuato la creazione di tre use cases. La velocità nella creazione dello use case è un importante fattore che fa capire, senza l'implementazione di un'interfaccia ma solo tramite codice, quanto sia facile crearne uno. Di seguito le tempistiche e le spiegazioni per esse:

- Metro (3 settimane circa): il primo use case creato e per il quale abbiamo speso più tempo, questo perché lo sviluppo ha comportato modifiche sostanziali al framework per rendere le azioni e gli stages di più facile creazione.

- ACLS (2 settimane): il secondo use case che abbiamo sviluppato, la parte più lunga nella creazione è stata la particolarità delle animazioni da creare, nel caso in cui avessimo avuto a disposizione già le animazioni il tempo si sarebbe ridotto di circa la metà.

- Parte conversazionale (2 giorni): il terzo ed ultimo use case, all'interno del quale abbiamo utilizzato già tutto ciò che il framework ci desse senza dover aggiungere qualcosa in particolare.

Riguardo le tempistiche e la facilità di implementazione siamo abbastanza soddisfatti dei risultati che abbiamo ottenuto.
Per quanto riguarda i vari dati raccolti abbiamo avuto riscontri positivi per quanto riguarda la velocità e le risposte del sistema, mentre alcuni risultati sul negativo per quanto riguarda l'habitability nel caso dell'ACLS e della naturalezza della conversazione nella parte conversazionale.
Le possibili risoluzioni e miglioramenti del framework:

- Ottimizzare meglio la gestione delle animazioni e le parti grafiche degli avatar in modo che si riesca ad ottenere un incremento degli fps.

- Integrare i movimenti all'interno dell'ambiente come parte accessoria dell'esperienza, in modo che l'interazione sia più varia e renda l'esperienza meno ripetitiva e noiosa.
- Velocizzare i tempi di creazione dell'audio con la rete neurale, in modo che la conversazione risulti più naturale.

Applicando questi miglioramenti l'esperienza risulterebbe molto più simile a possibili interazioni con gli esseri umani.

Elenco delle figure

Listings

Bibliografia

[1] Adaeze Adigwe et al. "The emotional voices database: Towards controlling the emotion dimension in voice generation systems". In: *arXiv preprint arXiv:1806.09514* (2018).

[2] Murray Campbell, A.Joseph Hoane e Feng hsiung Hsu. "Deep Blue". In: *Artificial Intelligence* 134.1 (2002), pp. 57–83. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00129-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370201001291.

[3] Justine Cassell. "Embodied conversational interface agents". In: *Communications of the ACM* 43.4 (2000), pp. 70–78.

[4] Homer Dudley. "The Vocoder—Electrical Re-Creation of Speech". In: *Journal of the Society of Motion Picture Engineers* 34.3 (1940), pp. 272–278. DOI: 10.5594/J10096.

[5] Howard E Gardner. *Frames of mind: The theory of multiple intelligences*. Hachette Uk, 2011.

[6] Eren Gölge et al. *coqui-ai/TTS: v0.1.2*. Ver. v0.1.2. Lug. 2021. DOI: 10.5281/zenodo.5075625. URL: https://doi.org/10.5281/zenodo.5075625.

[7] F. Itakura. "Line spectrum representation of linear predictor coefficients of speech signals". In: *The Journal of the Acoustical Society of America* 57.S1 (1975), S35–S35. DOI: 10.1121/1.1995189.

[8] Keith Ito e Linda Johnson. *The LJ Speech Dataset*. https://keithito.com/LJ-Speech-Dataset/. 2017.

[9] Mina A Khoei et al. "SpArNet: Sparse Asynchronous Neural Network execution for energy efficient inference". In: *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 2020, pp. 256–260. DOI: 10.1109/AICAS48895.2020.9073827.

[10] S. King e Vasilis Karaiskos. "The Blizzard Challenge 2013". In: 2014.

[11] Joshua Lederberg et al. "Applications of artificial intelligence for chemical inference. I. Number of possible organic compounds. Acyclic structures containing carbon, hydrogen, oxygen, and nitrogen". In: *Journal of the American Chemical Society* 91.11 (1969), pp. 2973–2976. DOI: 10.1021/ja01039a025.

[12] Elizabeth D Liddy. "Natural language processing". In: *Encyclopedia of Library and Information Science* (2001).

[13] John McCarthy et al. "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955". In: *AI Magazine* 27.4 (2006), p. 12. DOI: 10.1609/aimag.v27i4.1904. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/1904.

[14] John McDermott. "R1: A rule-based configurer of computer systems". In: *Artificial Intelligence* 19.1 (1982), pp. 39–88. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(82)90021-2. URL: https://www.sciencedirect.com/science/article/pii/0004370282900212.

[15] Shunji Mori, Hirobumi Nishida e Hiromitsu Yamada. *Optical character recognition*. John Wiley & Sons, Inc., 1999.

[16] Aäron van den Oord et al. "WaveNet: A Generative Model for Raw Audio". In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[17] Paul Resnick e Hal R Varian. "Recommender systems". In: *Communications of the ACM* 40.3 (1997), pp. 56–58.

[18] David E Rumelhart, Geoffrey E Hinton e Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[19] Alexander Serenko, Nick Bontis e Brian Detlor. "End-user adoption of animated interface agents in everyday work applications". In: *Behaviour & Information Technology* 26.2 (2007), pp. 119–132. DOI: 10.1080/01449290500260538.

[20] Pierre Sermanet et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint arXiv:1312.6229* (2013).

[21] Jonathan Shen et al. "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions". In: *CoRR* abs/1712.05884 (2017). arXiv: 1712.05884. URL: http://arxiv.org/abs/1712.05884.

[22] Sebastian Starke et al. "Neural State Machine for Character-Scene Interactions". In: *ACM Trans. Graph.* 38.6 (2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356505. URL: https://doi.org/10.1145/3355089.3356505.

[23] Dolça Tellols et al. "Enhancing sentient embodied conversational agents with machine learning". In: *Pattern Recognition Letters* 129 (2020), pp. 317–323.

[24] Dolça Tellols et al. "Sentient embodied conversational agents: architecture and evaluation." In: *CCIA*. 2018, pp. 312–321.

[25] Alexander Toshev e Christian Szegedy. "Deeppose: Human pose estimation via deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1653–1660.

[26] Brian C Vickery. "Knowledge representation: a brief review". In: *Journal of documentation* (1986).

[27] Nanyang Wang et al. "Pixel2mesh: Generating 3d mesh models from single rgb images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 52–67.

[28] Yuxuan Wang et al. "Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis". In: *CoRR* abs/1803.09017 (2018). arXiv: 1803.09017. URL: http://arxiv.org/abs/1803.09017.

[29] Yuxuan Wang et al. "Tacotron: A Fully End-to-End Text-To-Speech Synthesis Model". In: *CoRR* abs/1703.10135 (2017). arXiv: 1703.10135. URL: http://arxiv.org/abs/1703.10135.