POLITECNICO DI TORINO

Tesi di Laurea Magistrale in
Ingegneria del Cinema e dei Mezzi di Comunicazione

Advanced Storyboard: generazione automatica di storyboard mediante il controllo diretto dei personaggi

Anno Accademico 2021/2022

Relatore
Prof. Andrea Sanna

Co-Relatore
Federico Manuri

Candidato
Marco Scarzello

Abstract

Nell'era della memorizzazione digitale possiamo davvero affermare che un'immagine valga più di mille parole. Ed è per questo motivo che, tra la sceneggiatura scritta di un film e la sua realizzazione sul set, le produzioni grandi e piccole non trascurano quasi mai una fase importante: lo storyboard, ovvero una pre-visualizzazione grafica delle inquadrature da girare, ciascuna correlata da una serie di informazioni. Questa fase permette di risparmiare tempo e denaro durante quelle successive, ed è oggi usata anche in molti altri contesti, come nello *user-centered design*.

Ma esiste un modo per risparmiare tempo e denaro anche durante la fase stessa, la quale richiede tipicamente di disegnare e colorare ogni inquadratura, a mano o con una tavoletta grafica. Con uno strumento di computer grafica 3D, infatti, è possibile dare vita ad un mondo immaginato e poi esplorarlo da qualunque angolazione, spostandosi al suo interno. L'applicativo sviluppato in questo progetto permette proprio ciò, ma si spinge oltre: dopo aver ricostruito un set virtuale utilizzando modelli già caricati, l'utente può avviare la simulazione e prendere il controllo dei personaggi, per dare vita alla sceneggiatura, e salvare le inquadrature nel momento desiderato, per comporre uno storyboard automatico. Può muovere liberamente la camera, posizionare le luci e gestire il tempo a suo piacimento. Le azioni che un personaggio compie attivano, se presenti, animazioni corrispondenti.

Ma un'immagine da sola, fuori contesto, può creare ambiguità o restare incomprensibile. È per questo motivo che uno storyboard, spesso, riporta anche la descrizione dell'inquadratura rappresentata. In questo applicativo, tali descrizioni sono generate in automatico attraverso un sistema di interpretazione di vari tipi di eventi.

In questa ricerca è stato infine affrontato un problema di carattere semantico: quali azioni può eseguire un personaggio? In quali stati si può trovare? La soluzione non può che essere quella di permettere all'utente di definirlo, e questo avviene tramite un'interfaccia grafica *user friendly* che permette di accedere alle strutture dati che contengono queste informazioni e modificarle a piacimento.

Con questa pipeline di lavoro un regista, uno sceneggiatore o qualunque figura della pre-produzione può mettere alla prova una sceneggiatura per verificarne l'efficacia, testare inquadrature e scenografie diverse, ed infine ottenere uno storyboard, anche in distinte versioni. Se quello che si vuole realizzare è un prodotto d'animazione 3D, inoltre, è possibile importare direttamente nell'applicativo i modelli che saranno utilizzati, con tutte le loro animazioni, ed ottenere uno storyboard potenzialmente molto fedele al prodotto finale.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Quello descritto in questa tesi è un progetto realizzato con il motore grafico di Unity 3D che ambisce alla generazione automatica di storyboard. In questo capitolo viene illustrata la struttura di uno storyboard, mettendo in luce gli aspetti sui quali l'applicativo realizzato vuole intervenire, e vengono esposti gli obiettivi di questo progetto. Viene poi dedicato un capitolo allo stato dell'arte, ovvero alle soluzioni già esistenti che affrontano un problema analogo, al fine di identificare i territori già esplorati e quelli inesplorati, prima di intraprendere la realizzazione dell'applicativo. Seguono i capitoli relativi alle tecnologie utilizzate e alla vera realizzazione del prodotto, per chiudere infine con l'analisi degli user test ed i possibili sviluppi futuri.

1.1 Previs

Un prodotto audiovisivo attraversa tipicamente, prima della sua realizzazione, una fase di pre-produzione, che comprende la stesura di un concept, la scrittura della sceneggiatura e, in molti casi, una forma di pre-visualizzazione [1]. La pre-visualizzazione, o *previs*, permette di avere un riscontro visivo di una sceneggiatura, per capire se sia efficace, ma porta con sé anche nuove informazioni, per esempio la posizione e i movimenti della macchina da presa. Spesso, la previs coincide con lo *storyboard*, ovvero una sequenza di disegni, corrispondenti alle inquadrature che verranno realizzate, con una serie variabile di informazioni aggiuntive, ad esempio una breve descrizione per ciascun disegno, come quello in figura 1.1. Lo storyboard permette quindi di realizzare uno studio delle inquadrature prima che la troupe si rechi sul set, fornendo un utile supporto al regista e permettendo di risparmiare tempo.

La realizzazione dello storyboard coinvolge figure di diversi reparti: il regista per lo studio delle inquadrature, il direttore della fotografia per lo studio della disposizione delle luci, elemento che viene spesso inserito negli storyboard, ma anche sceneggiatori e production designer.

Va evidenziato inoltre come lo storyboard sia fondamentale, allo stesso modo, anche per i prodotti di animazione, nei quali sono coinvolti ulteriori reparti: modellazione, animazione, rigging, lighting, shading, rendering.

Possono far parte della previs anche altri elementi di supporto, oltre allo storyboard:

- Concept art o artwork, ovvero disegni di alta qualità che definiscono personaggi, pose, espressioni facciali;
- moodboard, ovvero compilation di immagini che suggeriscono il tipo di atmosfera che si vuole creare;
- animatic, ovvero video di animazione di scarsa qualità che riproducono una prima "versione" di quello che sarà il prodotto finale, che si tratti di un prodotto di animazione o in live action;
- shot list, un'elaborazione successiva dello storyboard in cui per ogni inquadratura vengono appuntati una serie di parametri come la focale della camera, il numero di take[1] previsti e varie durate;
- floor plan, ovvero una visione dall'alto di una scena, che permette di studiare, come in una pianta, la posizione dei personaggi, delle luci, delle camere, ed i movimenti di queste.

Lo storyboard, insieme eventualmente all'animatic, rimane comunque lo strumento più prezioso per la pre-produzione.

[1] Singole riprese

Figura 1.1: Semplice storyboard di una scena, composto solo da disegni abbozzati e brevi descrizioni [2]

1.2 Obiettivo della tesi

Gli storyboard possono essere realizzati con disegni più o meno curati: se si sceglie di mantenere un'alta qualità, l'impatto sul budget può essere significativo: uno storyboard artist può avere un costo unitario di 100 dollari a tavola.

Il progetto illustrato in questa tesi vuole fornire uno strumento user friendly per la realizzazione automatica di storyboard, che permetta la costruzione dell'ambiente 3D di una scena con un preset di modelli e la successiva simulazione della scena, all'interno di tale spazio 3D, con la possibilità di muovere la camera e salvare l'inquadratura quando la si ritiene soddisfacente, trasformandola nel disegno di uno storyboard. I vantaggi possono essere molteplici. Il più importante, ovvero quello relativo al tempo impiegato, è garantito dal fatto che ogni tavola non deve essere ridisegnata, poiché la scena 3D viene ricostruita soltanto una volta. Lo spazio 3D, inoltre, permette di studiare perfettamente la posizione degli oggetti, dei personaggi e delle luci, nonché i movimenti di camera e la prospettiva.

Tra le funzionalità che il progetto offre, è da menzionare la generazione automatica della descrizione che viene riportata sotto ogni vignetta, vista come uno degli obiettivi innovativi principali. Il progetto affronta, infine, il problema complesso della gestione dello stato dei personaggi e degli altri elementi di scena.

1.3 Elementi di uno storyboard

Non esiste un template definitivo, ma in base alle esigenze uno storyboard può essere più o meno arricchito di informazioni (si veda ad esempio la figura 1.2). In ordine di importanza, si possono trovare [3]:

- Vignetta: riporta il disegno, anche solo abbozzato, dei principali elementi dell'inquadratura; in versioni particolarmente scarne, gli oggetti possono anche essere sostituiti da scritte;
- Descrizione: un riassunto di quanto accade nella vignetta rappresentata;
- Dialoghi: eventuali dialoghi presenti nella scena;
- Numero del take: numero progressivo dell'inquadratura; per quanto possa sembrare inutile, diventa molto importante quando viene referenziato nella shot list, in cui l'ordine con cui si sceglie di girare le inquadrature non è solitamente lo stesso dello storyboard, che, invece, segue l'ordine diegetico[2];
- Tipo di obiettivo: corrisponde alla lunghezza focale ed incide sull'angolo di ripresa della camera;
- Durata: una stima della durata, in secondi, dell'inquadratura. In media, un'inquadratura dura 8 secondi;
- Aspect ratio: rapporto tra le dimensioni (orizzontale e verticale) dell'inquadratura, tipicamente di 1.85:1 nel cinema e di 16:9 (nuovo standard) nella televisione e in molti altri contesti;
- Tipo di inquadratura: campo (lungo, medio, totale...) nel caso in cui siano inquadrati elementi paesaggistici o oggetti inanimati, figura (mezza, intera, primo piano, primissimo piano, particolare...) nel caso in cui siano inquadrati esseri umani o umanoidi;
- Movimenti di camera: possono essere indicati con una scritta (pan, tilt, zoom, dolly, carrellata...) o con una freccia vuota disegnata direttamente sulla vignetta, che riproduce all'incirca il movimento da effettuare;
- Movimenti dei personaggi: talvolta indicati da una freccia piena direttamente sulla vignetta;
- Location: la stessa riportata sulla sceneggiatura (esterno/interno giorno/notte);

[2]Riferito all'universo della storia rappresentata nel prodotto audiovisivo.

- Sound FX: eventuali effetti sonori presenti nell'inquadratura;
- Altezza: espressa in centimetri, indica l'altezza della camera dal suolo;
- Costo dell'inquadratura: parametro raramente presente.

Figura 1.2: Due vignette di uno storyboard che riportano molte informazioni

Uno storyboard può trascendere la sua tipica struttura a vignette, talvolta, ed essere rappresentato con un solo disegno esteso, che coinvolge tutto l'ambiente di interesse e che riporta il movimento dei personaggi e della camera al suo interno, soprattutto nel caso in cui l'inquadratura sia una sola, ovvero un piano sequenza.

Questo tecnica di visualizzazione, infine, può anche essere utilizzata in contesti diversi da quelli del mercato audiovisivo, come per esempio nel design, al fine di rappresentare contesti d'uso di strumenti o interfacce grafiche. Uno studio di Truong e Abowd [4] identifica, in tale ambito, nuove variabili emerse dopo aver condotto dei test di realizzazione di storyboard su un campione di designer novizi, che hanno utilizzato anche strumenti digitali come Adobe Photoshop o Adobe Illustrator. Tra queste variabili si trovano il livello di dettaglio dei disegni, l'inclusione del tempo, la rappresentazione delle emozioni dei personaggi e la gestione del testo: su quest'ultima, risulta interessante l'utilizzo, talvolta, di nuvolette per i dialoghi o per i pensieri dei personaggi, inseriti direttamente nella vignetta, in uno stile di rappresentazione che si avvicina molto a quello del fumetto, e che tipicamente non si trova negli storyboard tradizionali. Tra i problemi evidenziati dai test sono emerse le difficoltà delle persone che non si ritenevano "capaci a disegnare", le quali hanno anche apprezzato le possibilità dei software di riutilizzare

la stessa immagine, effettuando solo qualche modifica, per realizzare una vignetta distinta, senza dover ridisegnare tutto da capo. Un'altra osservazione importante riguarda l'utilizzo del testo, giudicato come molto utile per comprendere la storia rappresentata: le sole immagini, spesso, forniscono una rappresentazione ambigua o addirittura incomprensibile. Emergono quindi proprio le due principali questioni che questo progetto vuole affrontare: quella della descrizione testuale e quella dell'automatizzazione delle vignette.

1.4 Cenni storici

Georges Méliès, celebre regista e illusionista francese, fu probabilmente il primo a utilizzare la tecnica dello storyboarding [5]. Walt Disney fu poi la personalità a portare in auge questa tecnica di previs, a partire dal 1930, trasformandola in uno standard dell'industria cinematografica. Da allora, la maggior parte delle produzioni cinematografiche passa attraverso questa fase, dalle pietre miliari dell'industria (figura 1.3) ai più recenti lungometraggi d'animazione 3D (figura 1.4). La tecnica dello storyboarding è oggi applicata anche nell'industria dei videogiochi, sia per cutscene[3] cinematografiche sia per rappresentare in poche vignette gli eventi principali del gameplay[4]. Più recentemente, anche nel campo dell'interaction design e dello sviluppo software gli storyboard vengono utilizzati per rappresentare casi d'uso ed esperienze utente.

[3]Sequenze non interattive, o semi-interattive, in cui il giocatore si limita ad osservare gli eventi.

[4]Flusso di gioco.

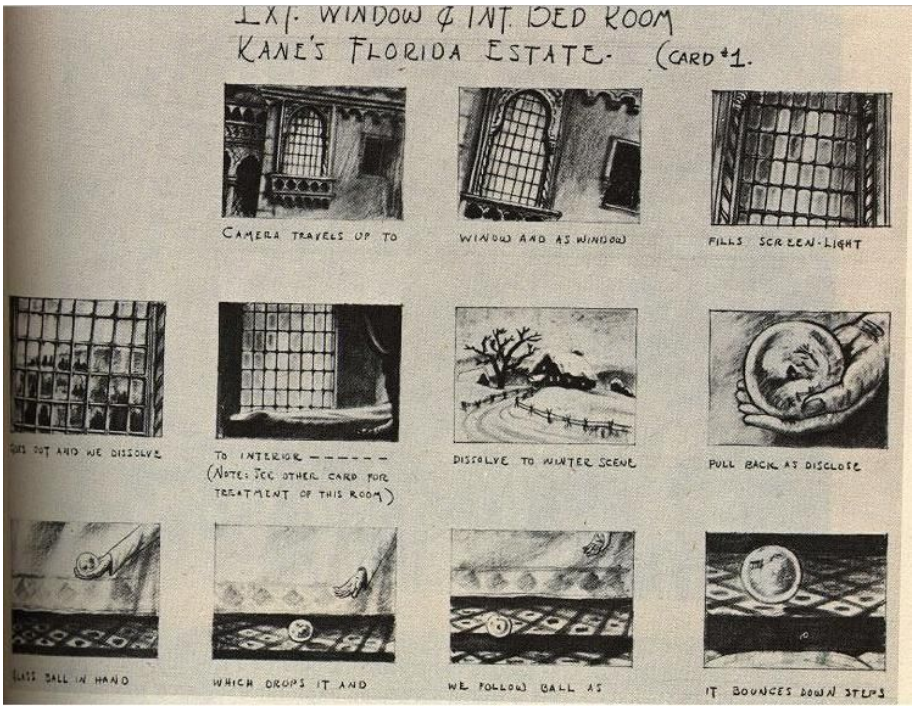


Figura 1.3: Orson Wells, per *Quarto Potere* (1941), fece realizzare storyboard molto dettagliati.

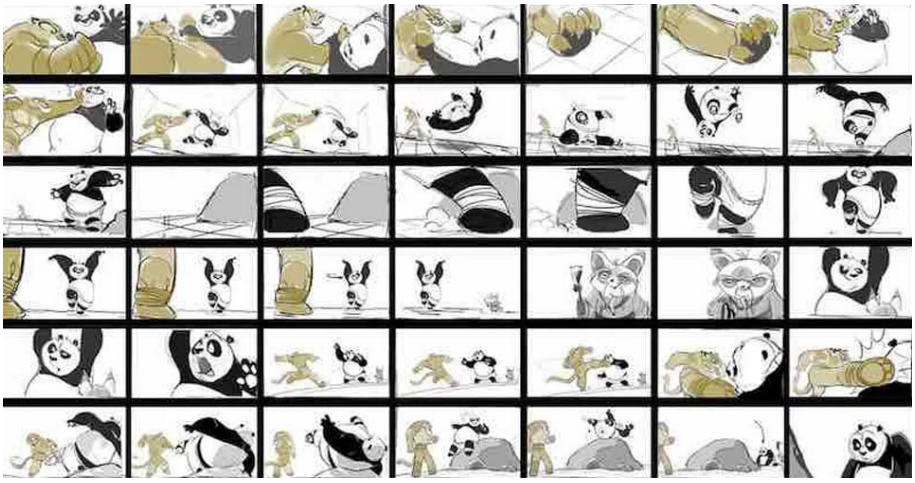


Figura 1.4: Storyboard ufficiale di *Kung fu Panda* (Dreamworks, 2008)

Capitolo 2

Stato dell'arte

In questo capitolo vengono analizzate soluzioni digitali esistenti per la realizzazione di storyboard e casi studio in cui sono presenti elementi che si direzionano verso i fini dell'applicativo. Si analizzano anche strumenti per la costruzione di ambienti 3D e la gestione di personaggi che si muovono e compiono azioni all'interno di essi, come nel caso di Project Spark. Viene infine introdotto il primo prototipo dell'applicativo da cui questo progetto è partito.

2.1 Generazione di storyboard

Si possono trovare online numerosi strumenti per facilitare la veloce creazione di storyboard, offrendo un'alternativa al disegno su carta. Molti di questi, tuttavia, offrono semplicemente dei template vuoti, in cui tutte le informazioni vanno inserite manualmente e nulla è automatizzato. A tal proposito si possono citare quelli messi a disposizione da Canva [6] e da Milanote [7].

2.1.1 Generazione non automatica

StoryboardThat è un sito web [8] che permette di utilizzare una vasta libreria di personaggi, oggetti e forme per costruire vignette 2D con logica a livelli, in cui ogni elemento è trasformabile attraverso rotazioni, scalamenti e traslazioni. In figura 2.1 sono riportati alcuni esempi di scenari ricostruibili. È possibile inserire i testi di descrizione delle vignette, nonché nuvolette con i dialoghi. Interessante è il grado di personalizzazione dei personaggi, che possiedono una finestra di editing in cui è possibile modificare la posizione delle loro varie parti del corpo, il colore dei singoli abiti o gli attributi del viso. Si tratta però di una soluzione ancora molto lontana dall'essere in grado di definire lo stato di un personaggio, o una sua azione.

Figura 2.1: Estetica ottenibile con StoryboardThat

Moltissimi altri tool, online o scaricabili, si posizionano allo stesso livello di complessità, ma hanno spesso un aspetto più professionale: tra questi, Studiobinder [9] e PanelForge [10].

2.1.2 Generazione automatica tramite linguaggio naturale

Nella letteratura scientifica si trovano poi strumenti più sofisticati, spesso ideati ad hoc per contesti di design, che permettono di ottenere storyboard "automatici". È il caso di *StoryDroid* [11], un tool capace di ricevere in input un'applicazione per Android e renderizzare un grafo, come quello in figura 2.2, che mostra tutte le possibili schermate dell'applicazione per ciascuna activity[1], con l'obiettivo di ridurre gli sforzi nella fase di ricerca per lo sviluppo di una nuova app, quando si analizzano i casi esistenti. È chiaro che il concetto di "storyboard", in questo caso, sia ispirato a quello a cui ci si riferisce in questa tesi, ma focalizzato su un altro contesto.

Senz'altro più interessante è uno studio di Baldwin e Ye [12] in cui viene proposto un sistema NLP[2] basato sul machine-learning per tradurre uno script in una visualizzazione grafica. Data una proposizione in input, anche articolata, il sistema estrae innanzitutto i verbi, e con l'utilizzo di un classificatore assegna a ciascun verbo una serie di comandi grafici necessari per la sua rappresentazione. La novità è che tali comandi grafici appartengono a un set generico, e sono automaticamente assegnati dall'algoritmo, senza essere in precedenza mappati ai verbi che li descrivono. I comandi grafici possiedono una serie di argomenti: il verbo (target), soggetti e complementi,

[1] Frammento base dell'interfaccia utente di un'applicazione Android.

[2] Natural Language Processing, ovvero la sottobranca dell'intelligenza artificiale che studia l'interpretazione del linguaggio umano da parte dei computer.

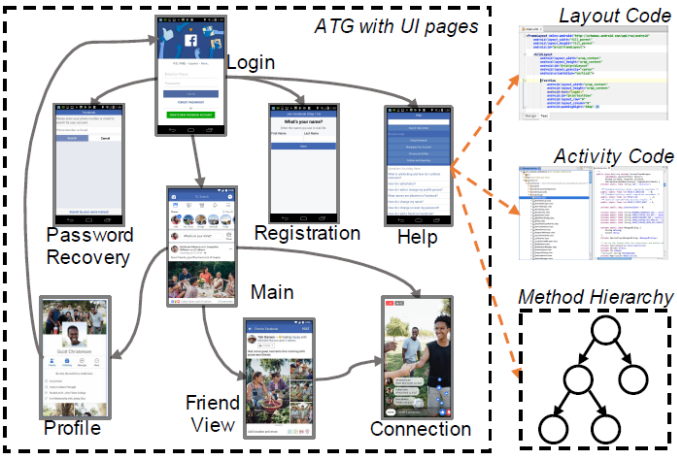


Figura 2.2: Storyboard di un'applicazione ottenuto con StoryDroid

argomenti di locazione. Per testare l'algoritmo ed ottenere una visualizzazione grafica delle azioni è necessario popolare a priori uno stage virtuale con gli elementi desiderati. Tali modelli vengono annotati con un WordNet synset, ovvero una descrizione semantica fornita dal database libero WordNet, di cui si parla nel capitolo 3 di questo documento. WordNet fornisce anche una lista di meronimi[3], che permettono di identificare sottoparti di un modello, come la mano di un personaggio: questo permette la generazione di scenari come quello della figura 2.3.

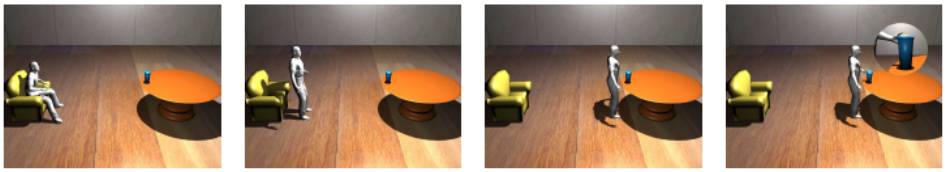


Figura 2.3: Visualizzazione della frase *The man grabs the mug on the table*

Simile è un sistema sviluppato da Disney [14], capace di produrre animazioni a partire da uno script ed inserirle in uno storyboard. Il sistema itera ricorsivamente una frase in input, scomponendola in proposizioni più semplici (figura 2.4), e viene affidato al game engine Unreal [13] il compito di pre-visualizzare tali azioni, utilizzando un set di 52 azioni predefinite e un set di modelli fornito.

[3]Sostantivi con relazione del tipo parte/sottoparte, come *pagina* e *libro*.

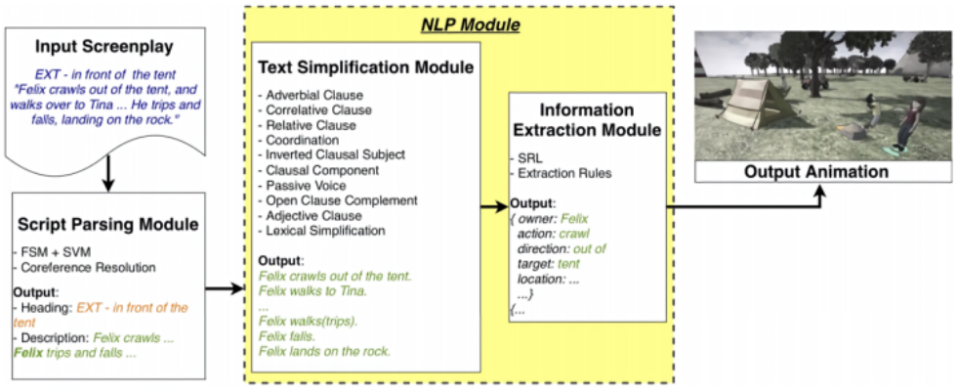


Figura 2.4: Un modulo di processamento del linguaggio naturale si basa sulla sua scomposizione e semplificazione.

2.1.3 Hitman: generazione da un preset

Un caso particolare degno di menzione è quello di uno studio [15] per la visualizzazione automatica della soluzione del livello di un videogioco tramite uno storyboard. La ricerca prende come caso di studio il videogioco Hitman [16] e, per la realizzazione della soluzione, utilizza un sistema chiamato Heuristic Search Planning, che attraverso una serie di iterazioni calcola la soluzione più veloce per raggiungere un determinato obiettivo, dato uno stato in input ed una serie di azioni di gioco disponibili. Tralasciando dettagli su questa fase e passando a quella di generazione dello storyboard, è stato utilizzato come base un database di sketch realizzati proprio per gli storyboard creati in fase di realizzazione del gioco Hitman. Questi disegni vengono selezionati automaticamente a partire dalla situazione descritta e montati fino a 3 layer nella stessa vignetta (attori, ambiente, atmosfera). Il processo è descritto nelle figure 2.5 e 2.6.

2.1.4 Generazione a partire da un video

Esistono infine casi meno rilevanti in cui uno storyboard viene ottenuto attraverso un algoritmo in grado di selezionare i frame più significativi o rappresentativi di una scena già girata. In uno studio di Microsoft [17], dal girato in input vengono estratti i frame significativi, viene effettuata una rimozione delle regioni irrilevanti dell'immagine (in modo da renderla più flessibile a nuovi contesti), seguita da una riunificazione dello stile (tramite filtri che rendono l'immagine cartoonesca). Un passaggio finale, da eseguire manualmente, consiste nel sostituire gli elementi rimasti con un render di modelli 3D. Un esempio di questo workflow è riportato nella figura 2.7. Dalla ricerca non emerge chiaramente l'obiettivo di questo sistema, se non quello di ispirare gli artisti a creare nuove storie partendo da storie esistenti.

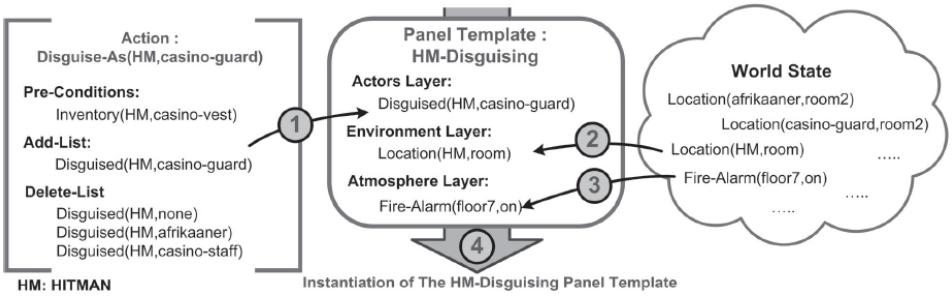


Figura 2.5: Nel primo step viene istanziata una descrizione di quella che dovrà essere una vignetta dello storyboard (al centro), a partire dall'azione e dalle condizioni del mondo. In questo esempio, la soluzione richiede che Hitman (HM) compia l'atto di travestirsi da guardia, quindi tale comando viene aggiunto all'actor layer.

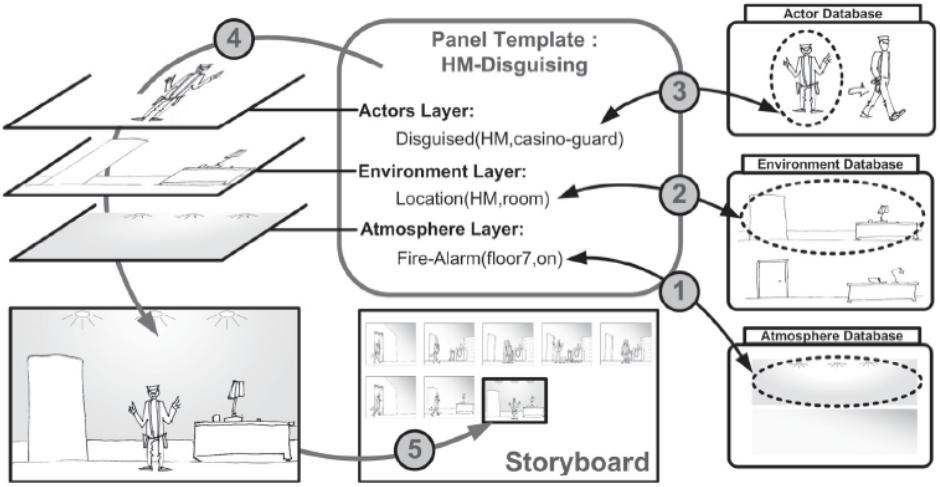


Figura 2.6: Nel secondo step il disegno viene effettivamente montato, a partire dalle informazioni precedentemente raccolte e dai database di disegni.

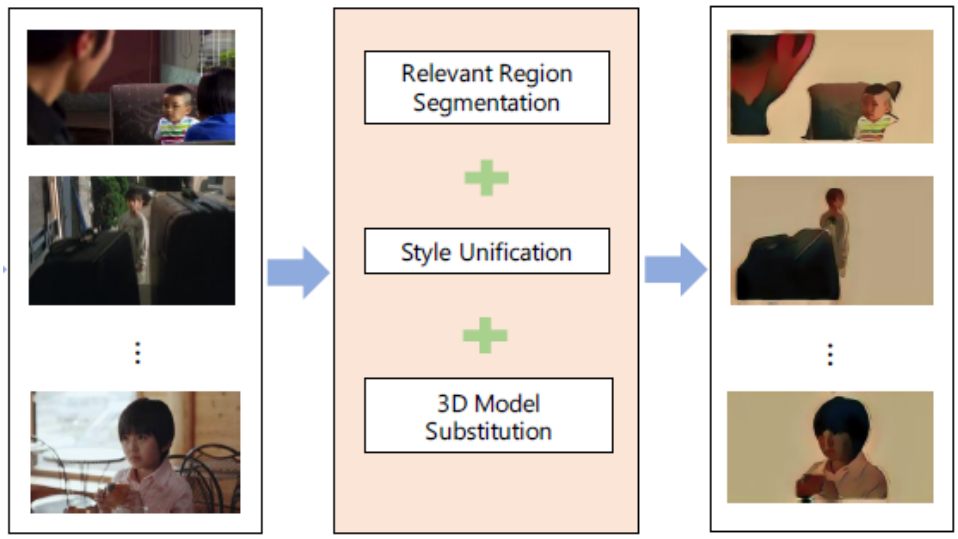


Figura 2.7: I tre passaggi effettuati sui frame di interesse

2.2 Generazione di descrizioni

Una delle funzionalità principali dell'applicativo descritto in questa tesi si pone in realtà sul verso opposto rispetto al trend finora esplorato, legato alla traduzione di testo in immagini. L'obiettivo è infatti quello di ottenere automaticamente la descrizione da assegnare alla vignetta dello storyboard, e dunque tradurre un'immagine in testo.

Esistono applicazioni che generano, tramite algoritmi di computer vision, la descrizione di un'immagine. Basti citare la tecnologia di Microsoft Edge [18] che assegna automaticamente, alle immagini che ne sono sprovviste, la descrizione per l'etichetta *alternative text*, visualizzabile quando il mouse si posiziona al di sopra di un'immagine, all'interno di un sito web. Gli algoritmi di questo tipo si basano su reti neurali capaci di estrarre le feature[4] di un'immagine e su altri modelli, come la Long Short-Term Memory network (LSTM), per la generazione della descrizione in linguaggio naturale, a partire dalle feature.

Molte informazioni, come quelle legate al tempo o ai rapporti di causa-effetto, vengono certamente perdute in un approccio di questo tipo. Occorre addentrarsi nel mondo dell'interattività per trovare casi in cui le azioni compiute e gli eventi accaduti vengono memorizzati e visualizzati tramite testo. Ciò accade, ad esempio, nei roguelike[5] come quello in figura 2.8, in cui la computer grafica è totalmente sostituita da caratteri testuali, utilizzati sia

[4]In computer vision, caratteristiche significative di un'immagine utili a risolvere un problema computazionale

[5]Giochi RPG che discendono dal famoso *Rogue*, con mappe procedurali ed interfaccia grafica basata su ASCII.

per visualizzare mappe ed oggetti, sia per descrivere ogni azione compiuta o evento esterno, altrimenti incomprensibile, lasciando grande spazio alla fantasia dell'utente.

Figura 2.8: Schermata di gioco di Nethack [19]: anche la mappa è rappresentata in caratteri ASCII, e al di sotto di essa si trova un log contenente ogni evento accaduto nella sessione di gioco, comprese le azioni dell'utente.

Si può infine citare il meccanismo del *diario* dei giochi per Nintendo DS Pokémon Diamante, Perla e Platino [20]: si tratta di un raccoglitore di informazioni salienti aggregate, relativo alle azioni diegetiche che il giocatore ha svolto nella sessione di gioco precedente.

2.3 Costruzione di ambienti 3D

Un requisito che l'applicativo descritto in questa tesi vuole soddisfare è quello di permettere all'utente la costruzione della scena, in cui vorrà successivamente realizzare una simulazione per ottenere le inquadrature di uno storyboard. Per questo motivo sono stati presi in esame due casi specifici: il primo è un gioco in cui, oltre a costruire l'ambiente, l'utente può programmare il comportamento dei personaggi e definire eventi, in uno strumento che nasce per permettere agli utenti di creare avventure con le loro regole; il secondo è un altro videogioco ideato apposta per il cinema, che permette di costruire un set virtuale.

2.3.1 Project Spark

Project Spark [22] è un framework per la creazione di videogiochi, che si avvicina al genere sandbox[6]. È stato pubblicato per Windows e Xbox One nel 2014, e non risulta più disponibile per l'acquisto dal 2016. La piattaforma era pensata per creare giochi e condividerli con altri utenti. Era dunque possibile scegliere se iniziare in un livello vuoto o testare le creazioni condivise da altri giocatori.

Una sessione di gioco su Project Spark si suddivide in due fasi, quella di creazione del mondo e quella di test. Durante la prima fase, detta *creator mode*, il tempo è fermo ed è possibile creare l'ambiente di gioco, attraverso l'interfaccia mostrata in figura 2.9, tramite l'utilizzo di 4 tool principali:

- Biome: fornisce un pennello che permette di creare un bioma in automatico, con tanto di elementi al suo interno come alberi e rocce, con generazione random;
- Paint: permette di pitturare il terreno con una texture a scelta;
- Sculpt: permette di modificare il terreno creando alture, fosse, erosioni, tramite un pennello con forma e dimensione personalizzabile;
- Prop: permette di posizionare nuovi oggetti nell'ambiente utilizzando menù a tendina.

Figura 2.9: Interfaccia di modellazione del terreno.

[6]Applicazioni in cui gli utenti possono modificare l'ambiente, creare livelli, eventualmente definire regole.

Il terreno viene modellato muovendo i pennelli con la tipica tecnica del trascinamento tramite mouse o tasti del joystick, e anche il posizionamento delle prop avviene con il drag and drop. Le prop, a differenza del terreno, possono essere programmate per avere un determinato comportamento. Esistono prop di diversi tipi:

- Characters: personaggi pensati per avere delle animazioni, per essere controllati dal giocatore o per ricoprire il ruolo di NPCs[7];
- Tumbling objects: oggetti sottoposti alle leggi della fisica, con cui i personaggi possono interagire causandone spostamento, caduta, rotolamento;
- Fixed objects: oggetti che non si possono spostare;
- Effetti: possono essere effetti visivi come aure e luci, o anche sorgenti sonore spazializzate[8]; contribuiscono al completamento del gioco.

È presente uno strumento di editing dedicato alle prop che permette di selezionarle per spostarle, eliminarle, riunirle in un gruppo chiamato assembly (nel quale verranno considerate come un'unica prop), eseguire un'operazione di attach (con cui una prop viene imparentata ad un'altra) ed altro ancora. Selezionando un personaggio, inoltre, è possibile modificare il suo aspetto con un'interfaccia tipica di tutti i giochi in cui le singole parti del corpo e i vestiti di un personaggio possono essere impostati, al momento della sua creazione. Su Spark è tuttavia possibile modificare anche parametri fisici e comportamentali come la velocità di movimento, il danno che verrebbe inflitto con un'arma (parametro di alto livello), eccetera.

Ogni prop possiede un *Brain*, un ambiente di pseudo-programmazione che permette di definire il suo comportamento attraverso quello che viene definito *kode* (rappresentabile come in figura 2.10), dal nome di kodu, un linguaggio di programmazione semplificato pubblicato da Microsoft FUSE Labs nel 2009. L'interfaccia del brain è costituita da coppie di `when` e `do`, concettualmente simili ad un if statement. I `when` e i do sono suddivisi in classi come movement, objects, appearance, sensors, controls. All'interno di ciascuna di queste classi sono raggruppati degli eventi che è possibile selezionare ed impostare sia come causa (`when`) sia come effetto (`do`). Tali eventi sono già programmati per avere una complessità piuttosto elevata. Per fare un esempio pratico, supponiamo di voler evidenziare un determinato oggetto quando il personaggio si trova nei dintorni di esso: è sufficiente impostare *detect* (dalla classe sensor) come `when` e *highlight* (dalla classe appearence) come `do`, senza dimenticare di impostare anche come complemento l'oggetto in questione, semplicemente aggiungendo sia al `when` sia al `do` un *in-world picker* (dalla classe objects) con cui si seleziona l'oggetto di interesse.

[7] Non-Player Character, personaggio non controllabile dall'utente.

[8] Audio stereo direzionato in base alla posizione relativa della sorgente.

Figura 2.10: Una linea di kode.

Per far muovere il personaggio è poi sufficiente impostare come `when` il sensore di interesse, ad esempio la levetta analogica sinistra di un joystick (dalla classe controller) e come `do` l'evento move (dalla classe movement), in grado automaticamente di comprendere la direzione in cui il personaggio deve muoversi. Altri eventi racchiudono poi comportamenti ulteriormente più complessi: l'evento *shoot* (dalla classe combat), ad esempio, fa espellere automaticamente una sfera di fuoco nella direzione in cui è rivolto il personaggio. Per personalizzare tale comportamento, è possibile associare allo shoot uno o più eventi della classe modifier. Le animazioni, infine, sono sempre automatiche, e già associate agli eventi che si programmano nel kode.

2.3.2 Cine Tracer

Cine Tracer [23] è un applicativo realizzato con Unreal Engine, disponibile sulla piattaforma Steam, che si pone ad un livello di complessità molto più alto. Dopo aver selezionato una mappa, l'utente si trova in un ambiente 3D semi-fotorealistico con la possibilità di accedere a vari menù per la costruzione della scena: prop, light, camera, actors, building. Moltissimi parametri degli oggetti, per esempio quelli riguardanti le luci, possono essere modificati per la massima personalizzazione dell'ambiente. Per quanto riguarda gli attori, i loro modelli possiedono un'armatura che permette di muoverli liberamente, facendo acquisire loro la posizione desiderata, ed implementano un sistema per realizzare animazioni. Da notare come dunque non siano in alcun modo programmati o programmabili, e non possano dunque realizzare azioni o gestire uno stato in automatico, in risposta ad altri eventi. Piazzando una camera (come in figura 2.11), poi, è possibile realizzare delle fotografie della scena e salvare le immagini per utilizzarle in uno storyboard: il gioco non offre un sistema per facilitarne la creazione. Molto dettagliato è invece il sistema di costruzione, a partire da oggetti elementari, di ambienti, muri, edifici: è quello tipico di un gioco sandbox.

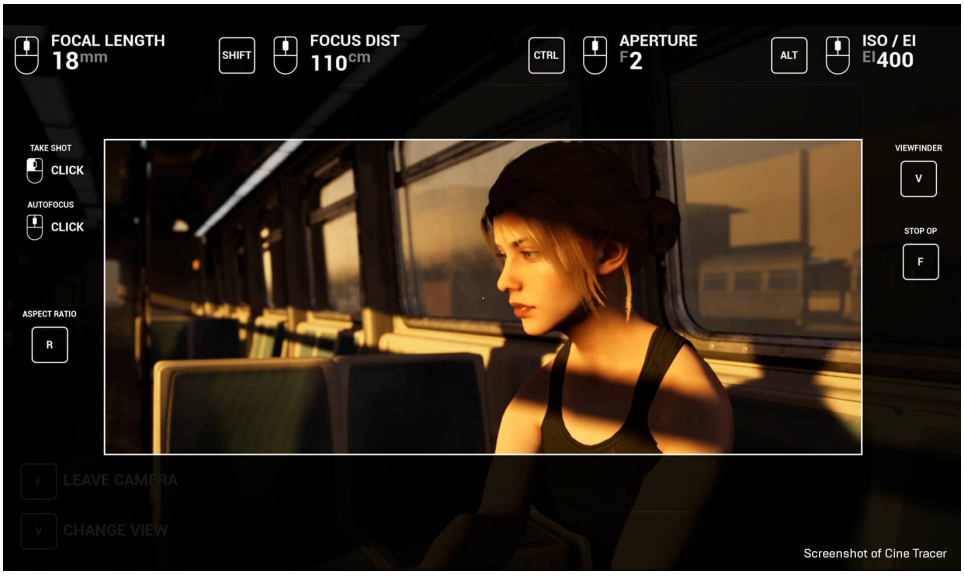


Figura 2.11: Inquadratura realizzata su un virtual set di Cine Tracer

2.4 Animatic (cenni)

Un animatic, come già menzionato, è una visualizzazione in sequenza delle vignette dello storyboard, che aggiunge una componente fondamentale utile a comprendere l'efficacia del ritmo di una scena: il *timing*. Un altro elemento che può essere presente è la colonna sonora provvisoria, o *scratch track*, anch'essa utile per effettuare valutazioni artistiche. Un animatic può essere costituito solamente dagli sketch dello storyboard, oppure riportare frame aggiuntivi. In alcuni casi, come quello in figura 2.12, può addirittura consistere in un girato d'animazione completo, di bassa qualità tecnica, ma molto utile per studiare punti macchina, movimenti e tempistiche, e accorciare di molto la durata delle riprese.

Figura 2.12: Comparazione tra il girato finale e l'animatic di *Dark Resurrection* [21]

Sul mercato si trovano innumerevoli strumenti per la realizzazione di

animatic, che di fatto coincidono con strumenti per la realizzazione di animazione frame by frame, in cui è sufficiente impostare un framerate di visualizzazione più basso, a piacimento. FlipaClip [24] è un esempio ricco di funzionalità: in quest'app gratuita per android è possibile realizzare frame con disegno a mano libera, elementi precostituiti o fotografie, impostare un framerate ed esportare il video. È anche presente una funzionalità per visualizzare una timeline in cui aggiungere tracce audio.

2.5 Punto di partenza: Applicativo Blender

L'applicativo descritto in questa tesi nasce dagli sviluppi futuri di un progetto embrionale che per primo ha affrontato il problema della realizzazione automatica di storyboard con gli stessi requisiti indirizzati in questa tesi [25]. Tale applicativo è stato realizzato con il software di computer grafica Blender utilizzando il Game Engine incorporato, che fornisce la possibilità di scipting in Python e la gestione degli eventi di gioco tramite *logic bricks*: sensori che ricevono dati in input, controllori che li elaborano (spesso tramite codice) e attuatori che modificano lo stato di gioco.

Il sistema, non pensato per essere generalizzato, è costituito da 3 casi d'uso preimpostati, il più topico dei quali coinvolge i personaggi di un cane e di una ragazza, all'interno di un canile. È possibile prendere il controllo di questi personaggi e della camera e muoverli nello spazio, facendo poi compiere loro delle animazioni (definite *actions*) dati certi comandi, ed effettuando uno screenshot per lo storyboard quando si desidera salvare l'inquadratura corrente, con affiliata la descrizione automatica degli eventi accaduti, come in figura 2.13. Al termine della simulazione, lo storyboard automatico viene visualizzato su un file html generato anch'esso automaticamente. La funzionalità più importante è quella di generazione automatica delle frasi, capace di interpretare soggetto, predicato e complemento di un'azione, e di aggiungere altre informazioni di contesto: il luogo in cui i personaggi si trovano, la contemporaneità di più azioni o il loro susseguirsi, la simultaneità di due azioni uguali compite da personaggi diversi, la vicinanza tra personaggi o tra personaggi e oggetti inanimati.

I principali elementi mancanti che hanno spinto alla continuazione di questo progetto, nonché ad una sua rivisitazione con un diverso approccio, sono: la mancanza di una libreria di modelli per la costruzione da parte dell'utente di un set virtuale personalizzato; la mancanza di generalizzazione delle azioni che i personaggi possono effettuare, da soli, su altri oggetti o su altri personaggi; la mancanza della gestione dello *stato* di personaggi ed oggetti, il cui concetto viene elaborato nel capitolo successivo.

Figura 2.13: Esempio dell'output finale dell'applicativo: vengono mostrati l'inquadratura scelta, la focale della camera, la location e la descrizione delle ultime azioni eseguite.

2.6 Stato di un personaggio

All'interno di un'applicazione interattiva, un personaggio può essere innanzitutto di due tipi: giocante (PG, ovvero controllabile dall'utente) o non giocante (NPC). In entrambi i casi, un concetto fondamentale che definisce la logica di comportamento dei personaggi è quello di *stato*. Lo stato di un personaggio si può rappresentare in modi diversi, a partire dalla schematizzazione di una macchina a stati finiti.

2.6.1 Diversi tipi di FSM

Una macchina a stati finiti [26], o *finite state machine* (FSM), è un automa[9] con cui è possibile descrivere o definire il funzionamento di molti sistemi. Tale modello si può rappresentare visivamente con un grafo[10]. Tale soluzione è facilmente applicabile per definire il comportamento di un NPC: si definiscono gli stati ed il comportamento che il personaggio assume in ogni stato, per poi definire gli eventi che determinano il passaggio da uno stato all'altro. Per fare un esempio molto basilare, la figura 2.14 rappresenta 4 possibili stati di un NPC: nello stato di *wander* verrà eseguito in loop il codice che porta il NPC a vagare nell'ambiente, presumibilmente seguendo dei percorsi predefiniti. In caso il giocatore sia vicino, il NPC passa allo stato di *attack*, ed eventualmente a quello di fuga se il giocatore contrattacca. Ritorna allo stato di wander se il giocatore si allontana. Se si trova in stato di fuga e i suoi punti salute sono bassi, invece, l'NPC passa nello stato *find aid*, che corrisponde presumibilmente ad un blocco di codice in cui il NPC avrà come target un oggetto in grado di ripristinare la sua salute. A determinare

[9] Sistema dinamico discreto e tempo-invariante.
[10] Insieme di punti - detti nodi - ed archi che li collegano, definendone le relazioni.

il passaggio da uno stato all'altro, dunque, sono variabili matematiche ben determinate come la salute o la distanza del giocatore, sempre monitorabili. Questa è la definizione più semplice possibile di intelligenza artificiale.

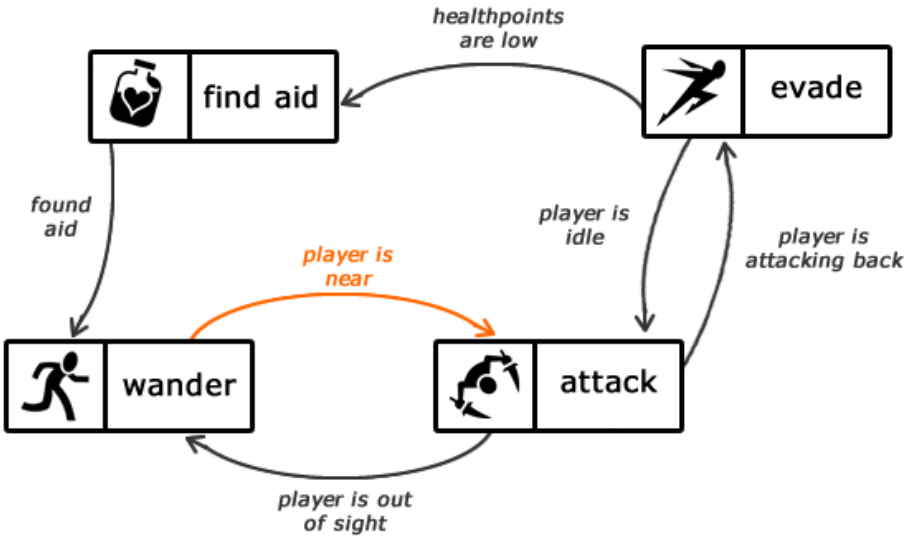


Figura 2.14: Semplice FSM di un personaggio non giocante

Esiste una versione tendenzialmente migliorata di una normale FSM, vale a dire una FSM basata su *stack*. Concettualmente, uno stack è una "pila" in cui i nuovi elementi vengono aggiunti al di sopra degli altri (con un *push*) oppure rimossi con un *pop*, considerando che non è possibile rimuovere elementi al di sotto di altri. In una FSM che risponde a questo modello, si identifica come stato attivo l'elemento in cima allo stack, ed il vantaggio è dato dal fatto che è possibile articolare una transizione in modi personalizzati, a partire da uno stato attivo C (si faccia riferimento alla figura 2.15:

- È possibile effettuare il pop dello stato C ed il push di un nuovo stato, come in una transizione di una normale FSM;
- È possibile effettuare il pop dello stato C lasciando che lo stato sottostante, ovvero quello precedente, diventi il nuovo stato attivo.
- È possibile effettuare il push di un nuovo stato facendo sì che, in caso di futuro pop del nuovo stato, lo stato attivo ritorni ad essere C.

Per quanto riguarda i personaggi giocabili, lo stato può essere senz'altro rappresentato con una FSM, ma all'aumentare della complessità questo modello diventa sempre più obsoleto. Deve infatti esistere, nel caso di un personaggio complesso, la possibilità di un multi-stato, ovvero evitare che

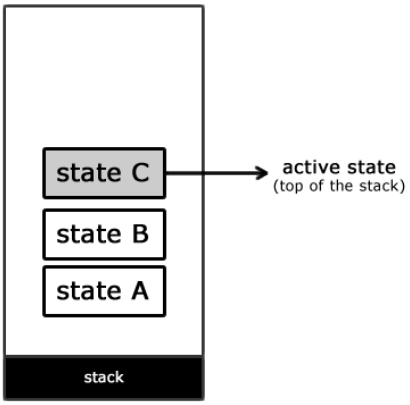


Figura 2.15: Rappresentazione di uno stack.

tutti gli stati siano esclusivi rispetto agli altri. Lo stato, inoltre, non servirebbe a definire il comportamento intelligente del personaggio, in quanto determinato dall'utente, bensì a rendere il sistema cosciente dello stato stesso e a definire le transazioni possibili da quel determinato stato.

Per definire un comportamento più complesso è possibile adottare l'approccio delle FSM di tipo gerarchico. Le *hierarchical state machines* (HFSM) si basano sul concetto di ereditarietà comportamentale degli stati: uno stato può essere definito come sotto-stato di un altro, aggiungendo un comportamento specifico. A tutti gli effetti, si applicano i principi della programmazione orientata agli oggetti: i sotto-stati funzionano grazie ai metodi degli stati principali e definiscono i loro nuovi metodi. Possono anche ridefinire metodi ereditati (polimorfismo).

2.6.2 FSM: formalizzazione matematica

Un sistema dinamico discreto e tempo-invariante è capace di comportarsi alla stessa maniera indipendentemente dall'istante di tempo in cui si agisce su di esso, e si definisce, appunto, automa. Un automa a stati finiti, spesso rappresentabile con un grafo di stati, si dice *deterministico* (ASFD) quando, per ogni coppia di stato e simbolo in ingresso esiste una ed una sola transizione allo stato successivo. Per una sua formalizzazione, si può definire Q come l'insieme dei possibili stati:

$$S = \{s_0, s_1, ..., s_n\}$$

I è invece l'insieme dei possibili simboli in ingresso, ovvero degli input che determinano una transizione:

$$I = \{i_0, i_1, ..., i_n\}$$

Analogamente, U è l'insieme degli stati in uscita:

$$U = \{u_0, u_1, ..., u_n\}$$

Si definisce poi la *funzione di transizione* δ:

$$\delta : I \times S \to S$$

Tale funzione esprime lo stato successivo in funzione dello stato attuale e dell'ingresso.

Un sistema che contempli più stati contemporanei si può rappresentare come un automa a stati finiti in cui gli stati di ingresso e di uscita sono rappresentati da una n-upla, dove n è il numero possibile di stati che l'automa può contemplare contemporaneamente. L'insieme S è in questo caso costituito dal prodotto cartesiano dei set possibili di stati, dove il prodotto cartesiano tra due set si può definire come:

$$A \times B = \{(a; B) \mid a \in A, b \in B\}$$

ovvero l'insieme di tutte le possibili coppie ordinate tali che il primo elemento della coppia appartenga al primo insieme, e il secondo elemento al secondo insieme.

2.6.3 Behavior tree

Quello del *behavior tree* [27] è un modello più complesso utilizzato nei personaggi dotati di intelligenza artificiale tipicamente non controllati dagli utenti. Per questo motivo, si offre qui soltanto un rapido cenno in merito alla loro struttura. Queste strutture ad albero, come quella in figura 2.16 sono costituite da nodi che, invece di rappresentare un semplice stato, rappresentano un compito (*task)* da eseguire. Esistono nodi di controllo di tipo *selector* o di tipo *sequence*: i primi selezionano il sotto-nodo da eseguire in base ad una condizione, gli altri eseguono in sequenza, da destra a sinistra, le operazioni indicate in tutti i sotto-nodi. L'esecuzione dei nodi è controllata da un *tick* inviato dalla radice, e i nodi ritornano un messaggio di *success, failure, running.*

2.6.4 Modellizzare lo stato

Lo stato di un personaggio controllato dall'utente, invece, è fortemente regolato dal tipo di gioco o applicazione in cui ci si trova: viene qui preso in esempio il personaggio giocante di *Fallout: New Vegas* [28], un Action RPG[11] post-apocalittico di tipo open-world[12]. Lo stato del personaggio può essere riassunto da alcune variabili principali:

[11]Role Playing Game, videogiochi in cui lo stile del personaggio giocante è altamente personalizzabile.

[12]Videogiochi in cui la mappa di gioco è un ambiente 3D da esplorare liberamente.

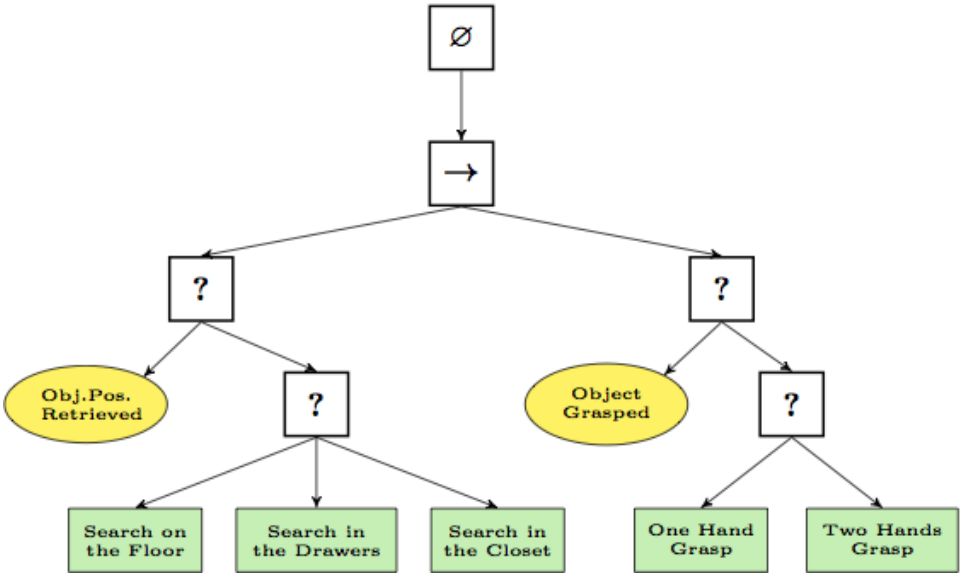


Figura 2.16: Rappresentazione ad albero del modello di intelligenza artificiale per la presa di un oggetto da parte di un robot a due braccia (→: nodo sequence; ?: nodo selector)

- I parametri di stato veri e propri come la salute, la fame, il livello di disidratazione, il livello di avvelenamento da radiazioni.
- Ciò di cui è in possesso il personaggio: gli oggetti nell'inventario, i vestiti che indossa in un dato momento, i soldi, l'arma corrente impugnata.
- Le abilità possedute dal personaggio, eventuali bonus, e la reputazione. Queste ultime variabili sono quelle che più di tutte determinano variazioni di comportamento quando il personaggio interagisce con altri NPC. In caso di reputazione bassa in un determinato gruppo, ad esempio, può essere attaccato all'istante.
- L'azione corrente che si sta eseguendo: nel caso in cui il personaggio sia seduto o coricato, dovrà alzarsi prima di poter effettuare qualunque altra cosa; nel caso in cui il personaggio stia intrattenendo un dialogo con un NPC, dovrà interromperlo per poter tornare a muoversi e sparare.

Proprio quest'ultimo punto si rivela essere il più interessante ai fini di questa tesi, poiché le azioni eseguibili dai personaggi controllati devono dipendere dal loro stato, e dallo stato dell'oggetto (se esiste) che subisce l'azione.

2.6.5 Generazione di diagrammi di stato

Prima di procedere all'implementazione del tool, interno all'applicazione, che permette di gestire i grafi di stato dei personaggi e degli oggetti, è stata svolta una ricerca sugli approcci esistenti per ottenere una struttura dati a partire da un grafo in stile UML[13] più user friendly, che sia modificabile dall'utente e in cui tali modifiche abbiano effetto sulla struttura dati sottostante.

Va menzionata innanzitutto l'esistenza di un linguaggio chiamato PlantUML [29], capace di eseguire il compito inverso: tradurre un semplice codice in un grafico automatico. Tra i tipi di diagrammi che supporta, si trova anche il grafo di una macchina a stati, come in figura 2.17.

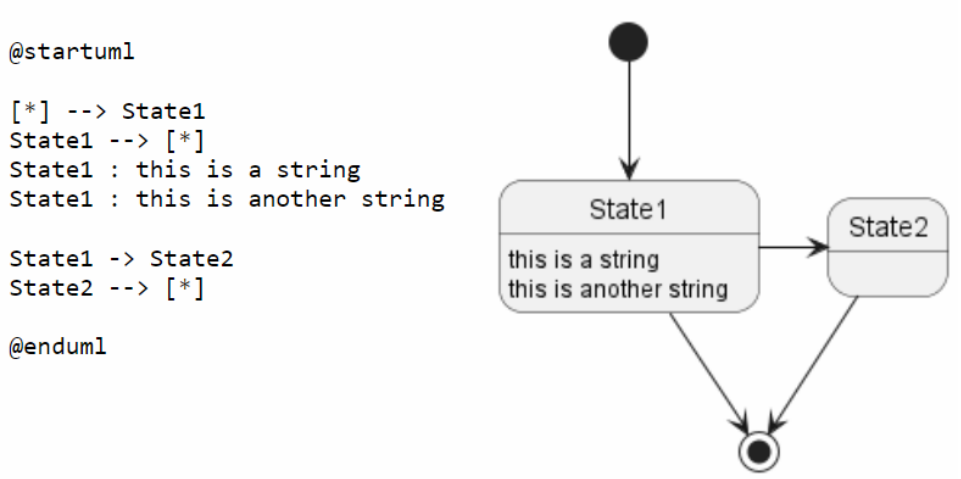


Figura 2.17: Diagramma generato con pseudo-codice PlantUML

I requisiti dell'applicativo descritto in questa tesi vogliono però che la struttura dati venga tradotta in un grafo e viceversa, mentre il linguaggio plantUML è monodirezionale e non permette di intervenire graficamente sul diagramma creato.

Stately [30] è un tool più complesso, pubblicato da XState, una libreria per JavaScript che permette di creare, interpretare ed eseguire macchine a stati finiti. Stately fornisce un editor in cui è possibile disegnare, direttamente sul browser, un diagramma di stato, e scaricare il codice corrispondente. Resta dunque uno strumento per programmatori, ma suggerisce l'idea del risultato che si mira ad ottenere in questo progetto. In figura 2.18 è riportato uno screenshot dell'interfaccia.

Un ultimo strumento testato è stato DRAKON [31], un linguaggio di programmazione visuale sviluppato nell'ambito del programma Buran, un progetto spaziale sovietico sospeso nel 1993. DRAKON formalizza la rap-

[13] Unified Modeling Language, protocollo per la visualizzazione di vari tipi di grafici pensato per l'ingegneria del software.

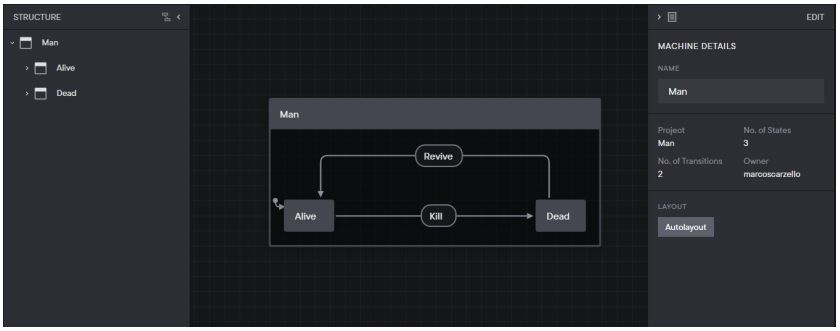


Figura 2.18: Interfaccia di stately.io

presentazione di flow chart e altri tipi di grafici, come i diagrammi di stato, attraverso determinate regole e macro-icone standard. Per fare un esempio, esiste il concetto di *skewer* ("spiedo"), secondo cui il percorso principale della diramazione di un flow chart deve essere composto da azioni allineate verticalmente lungo un segmento rettilineo. L'editor scaricabile dal sito ufficiale permette, infine, di tradurre automaticamente alcuni tipi di grafico in alcuni linguaggi di programmazione. L'opzione per tradurre un diagramma di stato in una macchina a stati finiti scritta in C# è attiva ma incompleta: genera la struttura scheletrica di una classe in cui si trovano soltanto gli attributi e alcuni metodi parziali.

2.7 Schema dei requisiti

Nella tabella 2.1 sono stati riassunti i punti di forza e debolezza dei casi analizzati in questo capitolo, secondo una serie di feature giudicabili come interessanti. I casi più ricchi di potenzialità, secondo questa tabella, risultano essere lo studio di Baldwin e Ye sulla generazione automatica di storyboard e Project Spark. L'obiettivo dell'applicativo sviluppato in questa tesi, tuttavia, si direziona nel verso opposto rispetto alla generazione di una scena 3D a partire dal linguaggio naturale: si vuole infatti ottenere la descrizione in linguaggio naturale di ciò che l'utente, liberamente, ha fatto accadere in scena. Project Spark, invece, permette una grande personalizzazione degli ambienti ed uno scripting visuale dei personaggi molto avanzato, ma non genera automaticamente storyboard e non fornisce alcuno strumento per farlo; si concentra inoltre sulla possibilità di assegnare comportamenti complessi ai personaggi per il design di un'avventura videoludica, ma non mette a disposizione un substrato per gestire animazioni, dizionari di azioni e altri aspetti semantici.

I requisiti innovativi dell'applicativo descritto in questa tesi risiedono pertanto nel supportare la costruzione personalizzata di un ambiente 3D in cui gli elementi eseguono automaticamente animazioni, corrispondenti alle

azioni selezionate dall'utente, ed il sistema interpreta queste azioni con una descrizione testuale, che aggiunge ad uno storyboard generato automaticamente, insieme ad altre informazioni. La semantica può essere ridefinita dall'utente con un sistema di gestione delle azioni e degli stati.

Feature	a	b	c	d	e	f	g	h	i	j
Grafica 3D			x	x				x	x	x
Animazioni				x					x	x
Generazione automatica		x	x	x	x	x				x
Natural language processing			x	x	x					
Collage con preset salvati	x				x	x				
Generazione di descrizioni						x	x			x
Costruzione stage virtuale			x					x	x	x
Ridefinizione eventi e azioni								x		x
Gestione luci e camere								x	x	x

Tabella 2.1: Feature più importanti enumerate nei casi studio analizzati

.a StoryboardThat

.b StoryDroid

.c NLP Automatic Storyboard (Baldwin, Ye)

.d Storyboard from screenplay (Disney)

.e Storyboard soluzioni livelli Hitman

.f Generazione da video (Microsoft)

.g Videogiochi roguelike

.h Project Spark

.i Cine Tracer

.j Obiettivi della tesi

Capitolo 3

Tecnologie utilizzate

In questo capitolo vengono presentati i framework e i plug-in principali utilizzati nello sviluppo del progetto. Il progetto è stato realizzato interamente con il motore grafico di Unity, ma prima di questa scelta alcune settimane sono state dedicate all'analisi delle potenzialità di altri framework, come il game engine di Blender.

3.1 Blender

Blender [32] è un software a licenza libera di modellazione, shading, texturing, rigging, animazione 3D, compositing e montaggio video. Supporta un motore di simulazione della fisica e possiede molte altre potenzialità, tra cui un game engine[1], scritto in C++ e capace di supportare lo scripting in Python [33], un linguaggio di programmazione di alto livello orientato agli oggetti. La logica dell'interfaccia di Blender si basa su finestre di editing manipolabili e scalabili a piacimento, che possono essere di diversi tipi, e in base alle esigenze il tipo di finestra può essere cambiato. Per lo sviluppo di un'applicazione interattiva, sono fondamentali le finestre di editing dei *logic bricks*, il *text editor* per lo scripting ed una *3D viewport* per visualizzare l'ambiente 3D in cui sono posizionati i modelli. Grazie all'interfaccia a "mattoni logici", ai modelli possono essere attribuiti rispettivamente:

- sensori, ovvero nodi di input che si occupano, in fase di simulazione, di ricevere informazioni da periferiche di input come la tastiera ed il mouse dell'utente, ma anche eventi in-game come collisioni tra modelli, variazioni di proprietà, raytracing;
- controllori, ovvero nodi intermedi che effettuano operazioni logiche a partire da determinati nodi di input, come semplici AND ed OR, o più

[1] Framework ideato principalmente per lo sviluppo di videogiochi e applicazioni interattive

complesse espressioni logiche, ed eventualmente funzioni contenute in uno script Python;

- attuatori, nodi di output che permettono di attivare movimenti, cambiamenti di proprietà e molti altri effetti.

Con questa interfaccia di programmazione a nodi, quindi, è possibile creare semplici applicazioni interattive anche senza ricorrere ad un file di codice. Lo scripting è però fondamentale per sviluppare applicazioni più complesse, e nel codice è possibile referenziare direttamente i logic brick di un modello, personalizzandone il comportamento.

Nelle versioni 2.8 e successive di Blender il game engine, chiamato fino ad allora BGE, è diventato incompatibile per scelta della Blender Foundation, ed un team di sviluppatori ha fondato una diramazione di Blender chiamata UPBGE (figura 3.1) [34], che di fatto non porta con sé nuove funzionalità, se non il fatto di rendere disponibile il game engine e restare al passo con gli aggiornamenti delle versioni di Blender canoniche. Nelle ultime versioni, UPBGE si distingue dalle vecchie versioni di BGE per l'utilizzo di nuovi nodi GLSL (OpenGL Shading Language) ed il miglioramento, in generale, delle funzionalità di lighting, nonché l'ottimizzazione di numerosi difetti.

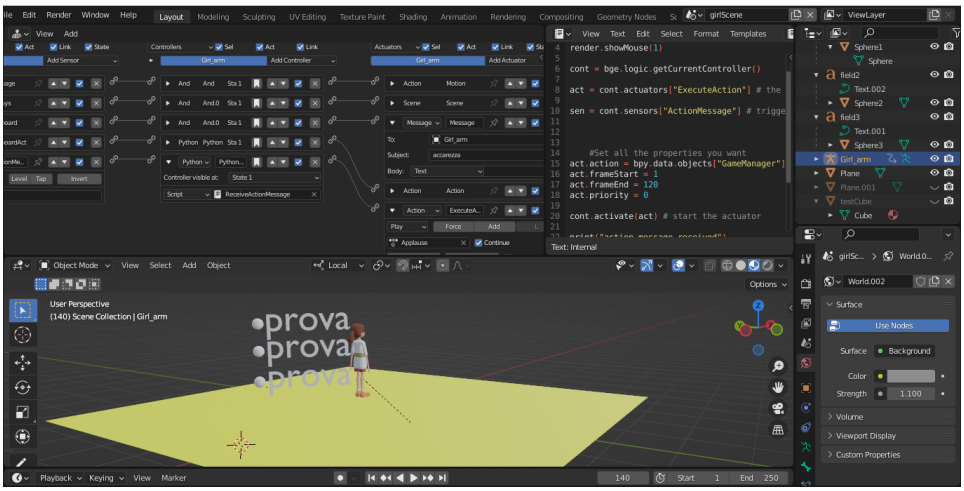


Figura 3.1: Interfaccia di UPBGE: le tre finestre principali utilizzate qui sono il logic bricks editor, il text editor e il 3D viewport.

Per questa tesi è stato sperimentato l'utilizzo di UPBGE per un'analisi di realizzabilità delle principali feature da includere nell'applicativo, ai fini di valutare la migliore scelta possibile del software. Il modello di un personaggio è stato posizionato su un piano e delle mesh[2] bidimensionali ottenute

[2]Maglia di triangoli che, in computer grafica, definisce la superficie di un oggetto tridimensionale nello spazio.

da un testo sono state utilizzate per l'interfaccia grafica che, in fase di simulazione, mostra automaticamente una lista delle azioni che il personaggio può compiere, ovvero le animazioni già preparate per il modello. Cliccando sul testo di un'azione, l'animazione corrispondente viene eseguita, ma per ottenere tale comportamento è necessario utilizzare numerosi sensori ed attuatori sulle scritte e sul modello, oltre a vari script. Il chiaro vantaggio di UPBGE è dunque quello di mettere a disposizione il medesimo ambiente per l'animazione tramite keyframe dei personaggi e la loro programmazione, ma risulta evidente come il framework sia pensato principalmente per applicazioni basate sulla grafica, in cui la componente logica non sia eccessivamente complessa. La mancanza di uno strumento per la creazione e la gestione di UI[3] ha portato infine alla scelta di Unity.

3.2 Unity

Unity [35] è un motore grafico a licenza EULA diventato ormai uno standard nella creazione di videogiochi indipendenti, semi-professionali e professionali, che si tratti di applicazioni per smartphone, desktop o console, nonché di realtà aumentata e realtà virtuale. Permette la realizzazione di applicazioni 2D o 3D. Tra le funzionalità degne di nota, supporta la compressione delle texture, le minimap e il multi-texturing e rende disponibili due distinte pipeline di rendering, la HDRP (High-Definition Render Pipeline) e la URP (Universal Render Pipeline). Rende disponibili numerosi package scaricabili, tramite il package manager, per integrare nuove funzionalità al progetto, mentre tramite l'Asset Store [36] è possibile scaricare asset di qualunque tipo, gratuiti o a pagamento, pubblicati da Unity o da utenti.

L'interfaccia di Unity, come mostrato in figura 3.2, è costituita da alcune finestre fondamentali:

- Scene: mostra l'ambiente 3D, in cui possono essere posizionati e trasformati i *GameObject*, ovvero qualunque oggetto di scena, come un modello, una camera, una sorgente di luce, o un elemento di UI come un bottone o un testo. Per lo stesso progetto si possono realizzare più scene, salvate come file con estensione .unity in un'apposita cartella. È possibile programmare la transizione tra scene;
- Hierarchy: mostra la gerarchia degli oggetti presenti in scena, che possono essere imparentati tra loro con un rapporto parent/child;
- Inspector: mostra i *component* che costituiscono il GameObject selezionato, e le loro proprietà. Tra i component si trovano spesso la *transform*, che definisce la trasformazione del GameObject nello spazio, il *renderer*, che ne definisce l'aspetto, mentre altri possono essere legati, ad esempio, alla simulazione della fisica o all'animazione.

[3] User Interface

Component specifici vengono introdotti nel capitolo successivo, ove necessario, quando verrà esplicato il loro utilizzo;

- Project: mostra la gerarchia delle cartelle che costituiscono un progetto. La cartella principale è chiamata Asset, e le best practices suggeriscono di organizzare i file importati (o creati) inserendoli in alcune sottocartelle principali come materials, textures, models, prefabs, animations, scenes, scripts. Anche i package importati sono costituiti da cartelle, automaticamente inserite sotto la radice;

- Game: finestra che viene messa in evidenza in automatico quando si preme il tasto play, testando l'applicazione, e mostra ciò che viene renderizzato real-time dalla camera impostata;

- Console: finestra fondamentale per il debug.

Altre finestre sono necessarie all'occorrenza, come quelle di animazione, di lighting, di rendering ecc.

Unity supporta molti formati di scambio per contenuti digitali, ma il più utile è senz'altro FBX, formato proprietario di Autodesk [37], supportato anche da Blender. Per trasferire un modello realizzato su Blender in un progetto Unity, ad esempio, è sufficiente effettuare un'esportazione in formato FBX ed inserire il file nella cartella models di Unity. Il modello può essere direttamente posizionato in scena, ma è anche possibile estrarre dal suo contenuto nuovi file, come ad esempio quelli dei materiali, per modificarli in Unity. È buona norma, tuttavia, utilizzare la logica dei *prefab* [40], ovvero trasformare ogni modello nel "prefabbricato" di un GameObject riutilizzabile e salvarlo in un'apposita cartella. Nel caso in cui un modello sia presente in varie istanze all'interno di una scena, sarà sufficiente modificare il prefab memorizzato e tutte le sue istanze subiranno le stesse modifiche. Anche quando si vogliono istanziare a runtime più copie di uno stesso modello occorre servirsi di un prefab. I prefab portano con sé numerosi vantaggi, perché possono contenere children e component. Inoltre, nuovi children e nuovi component possono essere applicati anche esclusivamente su una particolare istanza di un prefab, in scena.

Per il progetto di questa tesi è stata utilizzata la versione di Unity 2020.3.10f1.

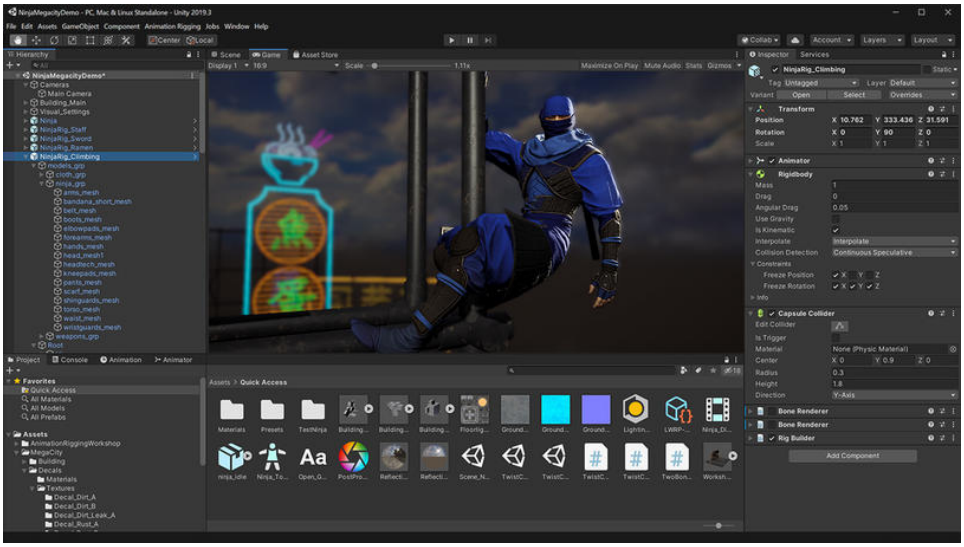


Figura 3.2: Interfaccia di Unity: l'inspector si trova sulla destra.

3.3 Visual Studio

Unity supporta lo scripting in C# [38], un linguaggio di programmazione multi-paradigma sviluppato da Microsoft nell'ambito dell'iniziativa .NET. Visual Studio [39] è invece un IDE[4] che supporta C# e che è possibile scaricare automaticamente insieme a Unity tramite Unity Hub, al momento dell'installazione, diventando di fatto l'ambiente standard di programmazione per Unity. Visual Studio utilizza un compilatore che converte il codice in un linguaggio intermedio chiamato IL (Intermediate Language), progettato per essere poi convertito in modo efficace in codice macchina su diversi tipi di dispositivi.

Al momento della creazione di uno script tramite Unity, questo viene automaticamente arricchito con le librerie necessarie ed eredita automaticamente dalla classe `MonoBehavior` della libreria `UnityEngine`, fondamentale per il funzionamento di uno script quando questo viene assegnato come component attivo di un GameObject. La libreria contiene alcune funzioni fondamentali. Le più importanti sono la funzione di `Start()` e quella di `Update()`. La prima viene chiamata al primo frame in cui il GameObject con lo script assegnato è attivo, quindi anche ogni volta in cui un nuovo GameObject viene istanziato runtime, e contiene solitamente inizializzazioni di ogni tipo. La funzione di `Update()`, invece, è chiamata ad ogni frame in cui il GameObject è attivo: deve contenere funzioni che si vogliono eseguire continuamente (spesso si tratta di controlli).

Per questo progetto è stata utilizzata la versione di Visual Studio 16.8.3.

[4] Ambiente di sviluppo integrato

3.4 WordNet

WordNet [42] è un database semantico-lessicale ideato presso l'Università di Princeton a partire dal 1995, che raggruppa set di vocaboli inglesi con significato affine, chiamati *synset*. Dal sito web [43] è possibile utilizzare questo database e scaricare package, oggi utilizzati principalmente in applicazioni di intelligenza artificiale per algoritmi di disambiguazione (WSD[5]).

La particolarità dei 117.000 synset forniti è che non si tratta soltanto di gruppi di sinonimi: esistono anche raggruppamenti effettuati in base a relazioni semantiche di tipo diverso, per esempio:

- Iperonimia/iponimia: un sostantivo X è iperonimo di Y se ogni Y è una specie di X, mentre per l'iponimia vale l'opposto;
- Olonimia/meronimia: un sostantivo X è olonimo di Y se ogni Y è una parte di X, mentre per la meronimia vale l'opposto;
- Troponimia/implicazione: un verbo X è troponimo di Y se nel fare l'attività X è necessario fare anche la Y, mentre è un'implicazione di Y se nel fare l'attività Y è necessario fare anche la X.

Questi vocaboli si possono visualizzare spesso attraverso grafi con collegamenti semantici, come in figura 3.3, ed il tool di ricerca sul sito di WordNet permette di esplorare questo database tramite hyperlink.

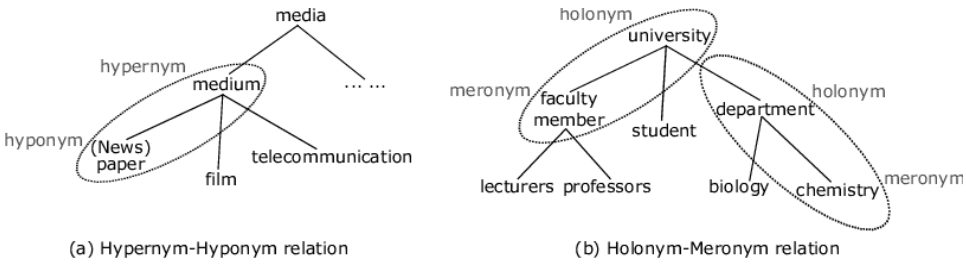


Figura 3.3: Esempi di relazioni tra vocaboli

Synthetic Intelligence Network (Syn) [44] è invece un brand di sviluppo e distribuzione di prodotti di intelligenza artificiale di proprietà di REVARN Cybernetics LLP. Tra i loro prodotti, hanno sviluppato e reso disponibile la libreria `Syn.WordNet`. È stato necessario invece scaricare manualmente i database contenenti i synset, e disporre tali file in una cartella in cui leggere da codice, per eseguire query nell'applicazione. Grazie alla libreria è possibile istanziare un oggetto di classe `WordNetEngine`, ed utilizzare il metodo `GetSynSets` per ottenere una lista di synset. Ciascun synset possiede attributi pubblici come `Words` (la lista di parole che contiene), `PartOfSpeech` e `Gloss`.

[5]Word Sense Disambiguation

In questo progetto, WordNet è stato utilizzato prevalentemente come dizionario dei sinonimi, ai fini di produrre variazioni casuali nelle frasi generate automaticamente, per renderle più simili al linguaggio naturale.

3.5 Componenti aggiuntivi

Per scaricare determinati pacchetti, come Syn.WordNet, è stato fondamentale installare un client in Unity chaimato NuGetForUnity [41], che fornisce l'interfaccia di un package manager per sviluppatori chiamato NuGet, sviluppato sempre da Microsoft per .NET Framework.

3.5.1 Pacchetti installati

Vengono qui elencati altri package fondamentali installati per il progetto:

- TextMeshPro [45]: soluzione molto efficiente per la gestione delle UI su Unity, in grado di offrire una qualità di rendering del testo molto superiore a quella degli strumenti predefiniti per la UI, oltre ad aggiungere e potenziare oggetti istanziabili come etichette, caselle di testo, bottoni, navbar, immagini, checkbox, menù a tendina ecc. Permette inoltre di personalizzare con ampio margine questi elementi;
- Cinemachine [46]: strumento per la gestione delle camere che semplifica di molto operazioni altrimenti eseguibili tramite complessi script, come il tracking di un target. Aggiunge funzionalità alle camere tradizionali e si interfaccia bene con gli strumenti di post-processing, ovvero di elaborazione dell'immagine già renderizzata;
- CsvHelper [47]: package installato tramite NuGet, nella versione 28.0.1, che offre un approccio semplificato per la gestione (lettura e scrittura) di file .csv[6] da codice.

Altri package ancora sono stati utilizzati durante la realizzazione del progetto ma non sono presenti nella versione finale, per motivi vari, ad esempio:

- Quick Outline: un pacchetto che fornisce un tool per aggiungere facilmente un bordo luminoso ai GameObject, per implementare un feedback di selezione. È stato deprecato, in questo progetto, in quanto ideato principalmente per la realtà virtuale e poco performante in caso di mesh complesse;
- GetSocialCapture: un progetto in grado di fornire la libreria necessaria per catturare in formato gif la registrazione dello schermo durante una sessione di gioco. Per l'applicativo finale, si è optato invece per

[6]Comma Separated Values, estensione per database testuali che raccolgono record in cui i campi sono separati da un carattere come la virgola

un animatic prototipale costituito semplicemente da una sequenza di immagini;

- Unity Recorder: uno package che fornisce una finestra capace di gestire la registrazione dello schermo in play mode, ma soltanto dall'editor di Unity, dunque non da parte di un utente che utilizzi soltanto la build finale dell'applicazione.

3.5.2 Librerie C#

Il seguente elenco riporta le librerie utilizzate in Visual Studio:

- `System`: namespace[7] che contiene molte sotto-diramazioni [48]. Tra quelle utilizzate:
  - `System.Collections` per gestire liste, array, dizionari, hashtable e altre collezioni di oggetti;
  - `System.Data` per gestire classi dell'architettura ADO.NET;
  - `System.IO` per leggere e scrivere file esterni e gestire le directory;
  - `System.Linq` per supportare query di tipo Language-Integrated Query (LINQ), utili per eseguire determinati metodi su collezioni di oggetti;
  - `System.Diagnostics` per interagire con i processi del sistema, i log eventi e i contatori delle prestazioni;
  - `System.Text` per supportare codifiche ASCII e convertire caratteri in byte.
- `UnityEngine`: namespace fondamentale per Unity [49]. Tra i suoi contenuti utilizzati:
  - `UnityEngine.UI` per la gestione di elementi dell'interfaccia utente;
  - `UnityEngine.SceneManagement` per la gestione di transizioni tra scene Unity; utente;
  - `UnityEngine.Windows.Speech` per l'input da microfono;
  - `UnityEngine.EventSystems` per la generazione di eventi tramite l'Event System.
- `UnityEditor`: API specifiche per l'editor di Unity;
- `TMPro`: libreria per la gestione dei componenti del package TextMesh-Pro;

[7] Collezione di classi a cui si può fare riferimento in un'altra.

- `Cinemachine`: libreria per la gestione dei componenti del package Cinemachine;
- `Syn.WordNet`: libreria per l'utilizzo del database WordNet.

Per ciascuna classe creata non sono state utilizzate tutte le librerie, ma soltanto quelle specifiche necessarie.

3.5.3 Asset esterni

Per gli oggetti di scena, nel progetto sono stati utilizzati modelli a licenza libera pubblicati online: è stato scaricato, ad esempio, un asset pack di base [50] per ottenere vari modelli di elementi di arredo interno per un'abitazione, mentre la maggior parte degli altri asset è stata ottenuta da Sketchfab [51], una piattaforma di proprietà di Epic Games per la compravendita e condivisione di modelli in vari stili e formati, talvolta con animazioni già integrate.

Mixamo [52] è invece una piattaforma di Proprietà di Adobe in cui sono resi disponibili vari modelli di umanoidi già sottoposti a rigging, ovvero provvisti di un'armatura interna che può essere utilizzata per animare il personaggio muovendone le componenti, dette ossa. Ma oltre ai personaggi, Mixamo mette a disposizione innumerevoli animazioni precostituite, che possono essere già visualizzate in anteprima sul sito web, assegnandole ad un personaggio a piacere, come mostrato in figura 3.4. È dunque possibile scaricare un file in formato FBX contenente un personaggio, un'animazione o un personaggio animato. I personaggi utilizzati nell'applicazione sono stati scaricati da Mixamo, e sono state assegnate loro alcune delle animazioni lì presenti.

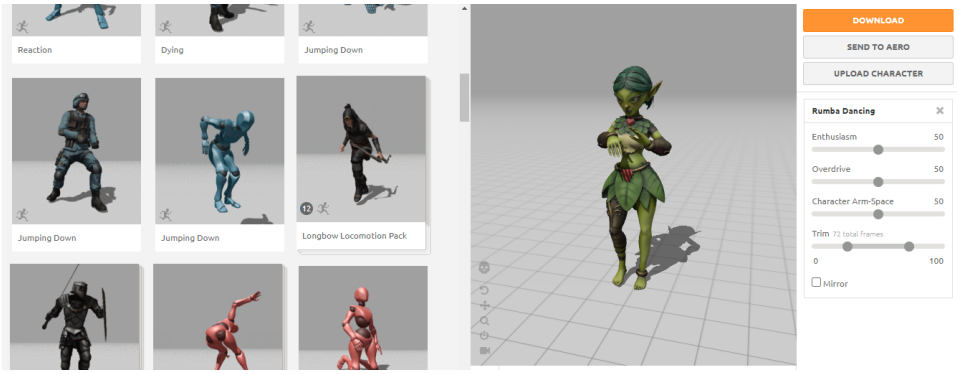


Figura 3.4: L'interfaccia di Mixamo

3.5.4 Risorse per il web

Il linguaggio JavaScript [53] è comunemente utilizzato nella programmazione web lato client ed integra un documento HTML[8], ovvero il contenuto statico di un sito web, scritto con un linguaggio di markup costituito da tag. Uno script Javascript può essere inserito in un file distinto rispetto a quello .html, oppure al suo interno, attraverso i tag `<script>`. Lo stesso principio vale per il CSS[9], ovvero la notazione utilizzata per assegnare uno stile ai vari elementi di un documento HTML.

Queste tecnologie sono state utilizzate per la realizzazione automatica della pagina web che presenta lo storyboard generato.

Il formato JSON[10], invece, è stato utilizzato per un file di testo che contiene informazioni sullo stato dei personaggi e degli oggetti: è stato scelto per il suo formato gerarchico nella memorizzazione di strutture dati e per la sua semplicità.

[8] Hyper Text Markup Language
[9] Cascade Style Sheet
[10] JavaScript Object Notation

Capitolo 4

Progettazione e realizzazione dell'applicativo

Riprendendo i requisiti emersi nella sezione 2.7 ed i ragionamenti sulla gestione dello stato di un personaggio, la fruizione dell'applicazione realizzata si distingue in tre fasi ben precise: quella di costruzione della scena, quella di simulazione/storyboarding e quella di modifica delle strutture dati relative agli stati ed alle azioni possibili, opzionale. La figura 4.1 rappresenta i moduli logici implementati. La maggior parte delle funzionalità si può sperimentare durante la seconda fase.

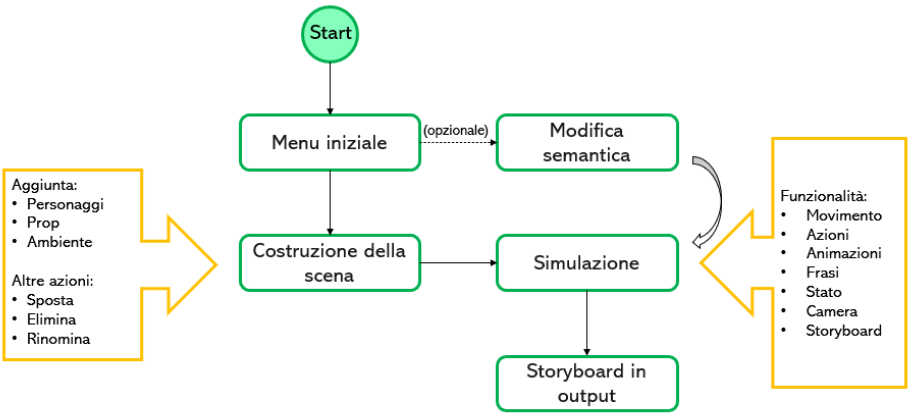


Figura 4.1: Pipeline della fruizione dell'applicazione

In questo capitolo si parte dalla modalità utilizzata per la gestione degli stati e delle azioni, ovvero la definizione della semantica di una simulazione.

4.1 Stati e azioni possibili

All'avvio dell'applicazione, l'utente si ritrova davanti ad un menù iniziale, come in figura 4.2, in cui è possibile scegliere se creare un nuovo stage virtuale, caricarne uno già creato oppure accedere all'interfaccia di modifica delle azioni e degli stati.

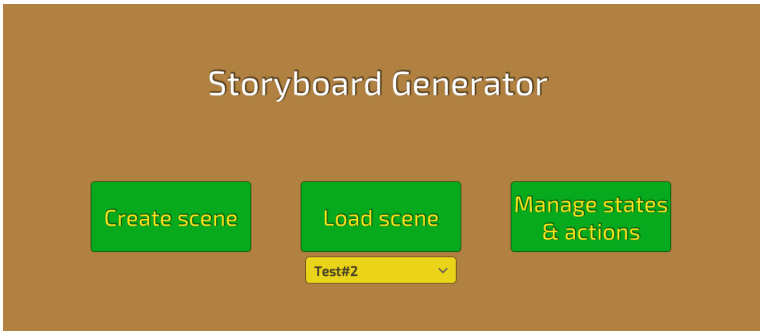


Figura 4.2: Il menù iniziale

Quello descritto in questa sezione è stato certamente il problema più complesso, che coinvolge la comprensione semantica di un contesto. Per quanto riguarda il concetto di stato, si pone il problema per cui un personaggio può eseguire una determinata azione su un oggetto in base allo stato di quest'ultimo: una *porta*, ad esempio, può essere aperta solo se è chiusa, e chiusa solo se è aperta, a l'azione di apertura o chiusura determina anche una transizione di stato. Altre azioni, come per esempio quella di dipingerla, possono essere eseguite sia su una porta aperta sia su una porta chiusa, ovvero indipendentemente dal suo stato, e non producono cambiamenti di stato. Non finisce qui: le azioni dipendono anche dal soggetto che le compie. Un cane non può dipingere una porta, ma possiamo supporre che sia capace di aprirla con una spinta, e magari non di chiuderla. Tutte queste informazioni vanno definite, ma risulta chiaro come non abbiano un carattere assoluto a causa della presenza di informazioni semantiche, e debbano quindi essere personalizzabili dall'utente in base alla sue esigenze.

Un ultimo problema da affrontare riguarda quello del multi-stato: una lavatrice, per esempio, può essere vuota o piena, ma anche accesa o spenta. Se l'essere piena implica che non sia vuota, non implica però che sia accesa, né spenta. Si tratta quindi di più stati indipendenti tra di loro. La lavatrice, in questo modello, possiede due stati che sono esclusivi solo con stati che appartengono al medesimo contesto semantico, come l'essere pieno/vuoto o l'essere acceso/spento.

Trattandosi di un problema semantico in cui la complessità può non avere limiti, infine, va detto che per realizzare un modello funzionante è necessario elaborare un compromesso definendo delle regole.

I requisiti sono quindi:

- Definire un dizionario con tutte le azioni che ogni personaggio può compiere, indipendentemente dagli stati, su ogni altro oggetto o personaggio, e su se stesso;
- Definire tutti gli stati in cui ogni elemento si può trovare e quali azioni determinano un cambiamento di stato verso un altro;
- Definire, per ogni stato, le azioni esclusive di tale stato;
- Permettere all'utente di ridefinire a sua scelta queste strutture, eliminando o aggiungendo stati e azioni.

4.1.1 Dizionario delle azioni

Dopo una prima soluzione deprecata, che coinvolgeva l'utilizzo di 4 file in formato CSV e non riusciva comunque ad affrontare tutte le casistiche possibili, è stata implementata una logica basata sull'utilizzo di due strutture di salvataggio per i dati: una tabella per il dizionario delle azioni ed un file in formato JSON per gli stati.

Partendo dalla prima soluzione, le azioni generiche possono essere rappresentate idealmente in un tabella a doppia entrata in cui ad ogni personaggio, per ogni altra prop o personaggio, si assegna una lista di azioni possibili. Devono qui essere incluse azioni semanticamente indipendenti dagli stati, ma anche quelle che dipenderebbero dallo stato. Possiamo definire questa struttura tabella *stateless* o tabella universale. Essendo ogni entrata una lista, si tratterebbe di una matrice tridimensionale, ma è possibile memorizzarla in un formato più semplice, il CSV, semplicemente elencando in ogni riga una lista di azioni e ponendo come primo elemento non un'azione, bensì una coppia personaggio-x, dove x è un altro personaggio o una prop. Anche le azioni che un personaggio può eseguire da solo si possono elencare in questa struttura, aggiungendo una riga con una parola chiave come "self" al posto di x, che viene interpretata dal codice. Un estratto di questa tabella è riportato nella tabella 4.1.

Questa struttura dati può essere modificata dall'utente nella scena di modifica delle azioni e degli stati, selezionando l'apposito bottone in alto, attraverso l'interfaccia user friendly mostrata in figura 4.3. L'interfaccia permette di selezionare un personaggio nella ScrollView[1] a sinistra, selezionare poi un complemento nella ScrollView centrale e visualizzare quindi le azioni possibili nella lista a destra: questa lista può essere modificata manualmente, e cliccando il tasto "save" viene aggiornato il file in memoria.

[1] Componente UI che permette di raccogliere i children con un layout di visualizzazione personalizzabile e di gestire barre di scorrimento laterali.

Soggetto-Complemento	Azioni
man-fireplace	light, extinguish, warm up by
cat-fireplace	warm up by
man-cat	pet, feed
cat-man	bite
man-man	talk to, greet, hug, attack
man-self	jump, smile, wait
cat-cat	cuddle, bite, clean, play with
cat-self	clean itself, jump
man-door	open, close, touch, break, repair
cat-door	scratch
...	...

Tabella 4.1: Esempi di azioni contenute nel file *actions.csv*. Per ogni possibile soggetto deve essere specificato ogni possibile complemento, compresi gli altri soggetti ed il soggetto stesso (con "self").

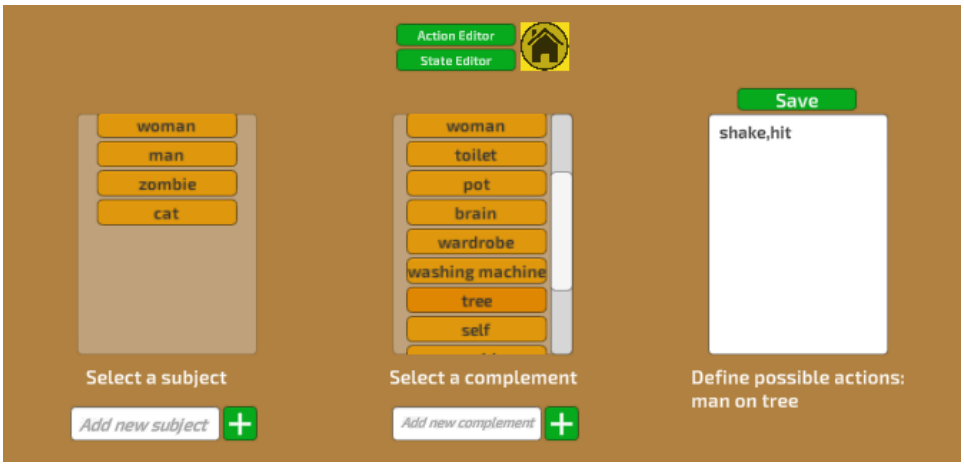


Figura 4.3: menù di modifica delle azioni

Il GameObject *Manager* di questa scena possiede il componente `Actions-Datatable` (script), che si occupa di questa procedura: per visualizzare tutti i personaggi attraverso bottoni imparentati alla ScrollView di destra viene invocato il suo metodo `LoadData()`, che legge (con la classe `StreamReader`) il file CSV ed istanzia un bottone per ogni distinto soggetto possibile. Ad ogni bottone attribuisce un listener che richiama il metodo `ShowObjects()`, il quale istanzia i bottoni per tutti i possibili complementi, inclusi quelli per i quali, nel file, non si trova ancora una linea dedicata al soggetto selezionato. `ShowActions(string character, string complement)` è il metodo che viene richiamato quando devono essere visualizzate le azioni nella casella di testo editabile. Premendo il tasto "save" viene invece invocato il metodo `SaveChanges()`, il quale aggiunge (o aggiorna) la linea specifica del file.

Tramite le caselle di testo sottostanti, invece, è possibile aggiungere alle liste bottoni corrispondenti a soggetti o complementi che non si trovano ancora nel file, per creare una nuova riga e definire le azioni da zero. I tasti di aggiunta invocano i metodi `AddSubject()` o `AddComplement()`. Se non si preme il tasto "save" prima di uscire dalla schermata, aggiungendo almeno una riga, i bottoni aggiunti vanno persi.

4.1.2 Memorizzazione degli stati

Gli stati degli elementi, ove presenti, sono stati memorizzati in un file di testo chiamato *states.txt*, utilizzando però il formato JSON, in cui sono presenti importanti attributi, per ogni oggetto. Un personaggio (o una prop) viene memorizzato con il proprio nome, una lista chiamata *transitions* che riporta ogni transizione di stato possibile nel formato `stato1>azione>stato2` ed una lista chiamata *S_A* che riporta, per ogni stato, la lista delle azioni esclusive di quel determinato stato, nel formato `stato1: azione1, azione2...` Per un esempio, si può fare riferimento alla figura 4.4.

```
{
    "name": "fireplace",
    "transitions": [
        "extinct>light>burning",
        "burning>extinguish>extinct"
    ],
    "S_A": [
        "burning:extinguish,warm up by",
        "extinct:light"
    ]
},
```

Figura 4.4: L'oggetto *fireplace* può trovarsi nello stato *extinct* o *burning*, e le azioni *light* e *extinguish* determinano il passaggio tra questi stati. Solo nello stato *extinct* è possibile effettuare l'azione *light*, e solo nello stato *burning* è possibile effettuare *extinct* e *warm up by* (ovvero riscaldarsi).

L'utente può modificare questo file, ancora una volta, dalla schermata di modifica delle azioni e degli stati, mostrata in figura 4.5, cliccando sull'apposito bottone in alto (*states editor*). Selezionando un oggetto o un personaggio dal menù a tendina in alto a sinistra, viene visualizzato un pannello dotato di una ScrollView in cui ogni stato è rappresentato con una *bolla* ed ogni transizione con una freccia. Si tratta, in entrambi i casi, di caselle di testo modificabili, attraverso cui l'utente può rinominare stati e transizioni. È possibile inoltre utilizzare le bolle vuote, in fondo, per aggiungere nuovi stati e, conseguentemente, nuove transizioni in altri stati. Se non si aggiunge una transizione che porta ad uno stato, esso viene comunque memorizzato correttamente, ma sarà inaccessibile. Per definire le azioni esclusive di ogni stato si può utilizzare invece la casella di testo sulla destra, che fa riferimento allo stato accanto a cui si trova. Il bottone "save" aggiorna effettivamente il file.

Vengono istanziate dal sistema anche delle barre orizzontali, utili a suddividere contesti semantici: nell'esempio mostrato in figura, questi contesti sono la caratteristica di una lavatrice di essere piena o vuota e quella di essere accesa o spenta. Serve dunque per visualizzare meglio il concetto di multi-stato.

Infine, è possibile aggiungere anche qui un personaggio o una prop che ancora non sono stati salvati nel file, utilizzando la casella di testo in basso a sinistra. Selezionando poi l'elemento dal menù a tendina, verrà mostrato un diagramma vuoto in cui è possibile aggiungere il primo stato.

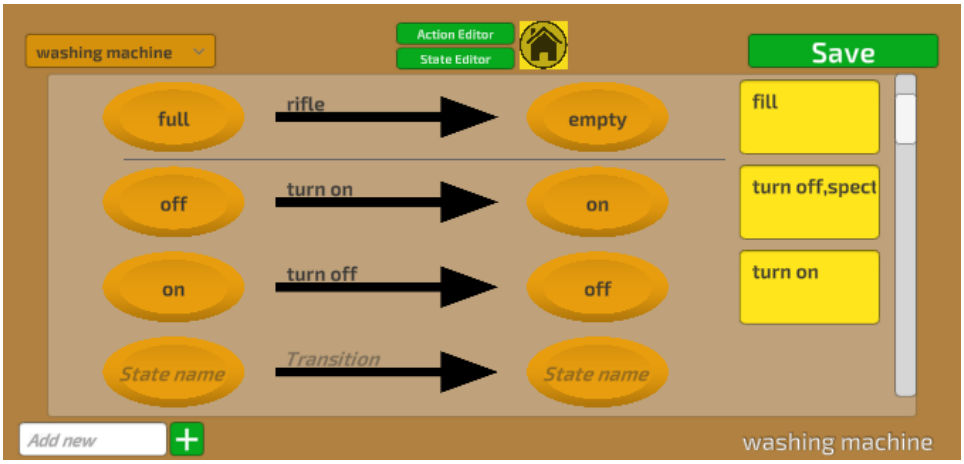


Figura 4.5: Editor degli stati

`StatesManager` è la classe che gestisce questo menù, attribuita sempre al GameObject Manager. Al suo interno, in realtà, sono contenute le definizioni di altre due classi, che servono a rispecchiare la struttura del file JSON: la classe `Model`, che rappresenta un oggetto del JSON e contiene dunque i tre attributi mostrati in figura 4.4, e la classe `ModelList`, che ha come

attributo una lista di `Model`. La libreria `UnityEngine` implementa la classe `JsonUtility`, che consente di chiamare il metodo `FromJson<ModelList>-(textJSON.text)` per salvare il file JSON in `ModelList`, dando in input il testo di un oggetto di tipo `TextAsset`, che deve referenziare il file.

Il metodo `InitializeDropdown()` inserisce ogni oggetto del JSON nel menù a tendina, e a quest'ultimo viene affidato un listener che, quando il valore della tendina cambia, deve mostrare il corretto `Model`, attraverso il metodo `DropDownObjectSelected()`: questo metodo rielabora le stringhe lette negli attributi dell'oggetto e si occupa di istanziare prefab delle bolle, delle frecce e delle caselle con liste di azioni, considerando anche la dimensione dello schermo per piazzare correttamente questi elementi di UI. Il metodo `WriteJson()` rielabora tutte le informazioni su schermo e ricostruisce il `Model` da salvare, aggiungendolo infine alla `ModelList` e trasformando quest'ultima in una stringa in formato JSON con il metodo `JsonUtility.ToJson()`, per poi scriverla nel file.

La classe contiene infine i metodi `TraceLine()` (per tracciare i separatori orizzontali) e `AddToJSON()` (per aggiungere oggetti non ancora presenti).

4.1.3 Funzionamento all'interno dell'applicativo

È possibile esplicare il funzionamento della classe `ActionsDataBase`, attribuita come component di GameManager nella scena principale, che fa riferimento alle strutture dati descritte. Questa classe legge i file di entrambe le strutture dati, salvando il file JSON con lo stesso meccanismo descritto in precedenza ed il file delle azioni con una hashtable. Il suo metodo fondamentale è `ReturnActions(string character, string complement, List<string> state, bool self)`, che restituisce la lista delle azioni che è necessario mostrare quando un personaggio interagisce con qualcosa. Questo metodo considera dapprima tutte le azioni possibili (associate ad una determinata coppia soggetto-complemento) lette dalla hashtable, poi itera ogni stato che riceve in input e determina, tramite il JSON degli stati, le azioni che vanno preservate (ovvero quelle tipiche di uno stato corrente) e quelle che invece vanno rimosse (ovvero quelle esclusive di uno stato non corrente). Le azioni che non dipendono dagli stati vengono lasciate. Un esempio finale di questa logica viene mostrato nella figura 4.6.

La classe `ActionsDataBase`, infine, implementa i metodi `GetPossible-States()` e `GetStateTransition()`, entrambi richiamati dalla classe `State` rispettivamente per: inizializzare lo stato di un elemento quando viene aggiunto in scena considerando i suoi stati di numero dispari (poiché il sistema implementa il concetto di multi-stato binario); ottenere il nome del nuovo stato se un'azione eseguita corrisponde ad una transizione di stato.

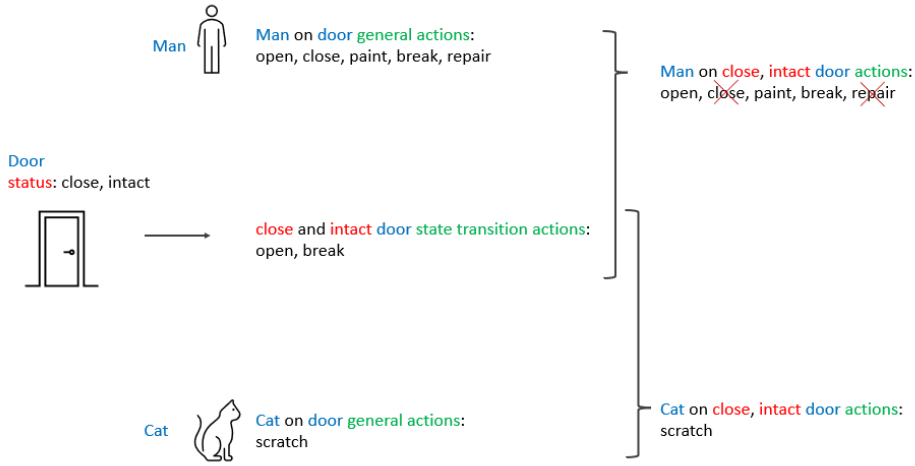


Figura 4.6: In questo esempio si può osservare nuovamente come le azioni generiche che un uomo è capace di eseguire su una porta vengano filtrate dallo stato della porta. Quelle che non dipendono da alcun tipo di stato della porta, come l'atto di dipingerla, vengono preservate. Più semplice è il caso del gatto.

4.2 Costruzione della scena

Dal menù iniziale, scegliendo l'opzione "Create storyboard", l'utente di trova davanti ad un mondo 3D vuoto in cui una UI mostra 3 menù a tendina (dropdown) dedicati rispettivamente alla costruzione dell'ambiente, al posizionamento di nuove prop e al posizionamento di nuovi personaggi, come in figura 4.7. Selezionando un elemento dal menù a tendina e cliccando sul rispettivo bottone "+" di aggiunta, l'elemento viene istanziato e segue la posizione del mouse. Per posizionarlo definitivamente nel punto desiderato, è sufficiente cliccare nuovamente. Si possono posizionare anche più GameObject uguali.

Un toggle riportante l'etichetta "Guide grid" può essere attivato a discrezione per facilitare l'allineamento degli elementi ad una griglia invisibile. Questa tecnica, oltre a permettere di disporre gli elementi in modo armonico, può essere utile a creare muri, pavimenti o altri elementi composti dalla ripetizione regolare dello stesso modulo. In ogni caso, se l'utente cerca di posizionare nuovi GameObject al di sopra di altri già presenti, fallisce. L'utente può premere il tasto "R" per ruotare di 90° il GameObject selezionato, se ha già premuto il bottone di aggiunta ma non ha ancora cliccato sul terreno per scegliere la posizione.

Sempre nella fase di costruzione, quando viene selezionato un GameObject già posizionato in scena, appare un nuovo menù di UI (panel), in alto a

destra, riportante il nome dell'elemento e 3 bottoni: move, delete, rename. Cliccando sul primo è possibile riposizionare il GameObject, cliccando sul secondo viene eliminato il GameObject selezionato e cliccando sul terzo il GameObject viene rinominato con il valore testuale presente nella casella di input soprastante. Il sistema memorizza sia il nome originale dell'elemento sia quello nuovo che gli è stato attribuito. È possibile rinominare non solo un personaggio, ma anche una prop: questo può essere utile nei casi un cui il modello esatto della prop desiderata non sia disponibile, e ne viene posizionato in scena uno simile. In uno storyboard, infatti, l'aspetto di un elemento non è prioritario quanto la sua dimensione o il suo posizionamento nello spazio.

Un altro toggle, vicino a quello della griglia guida, permette di cambiare l'illuminazione della scena da diurna a notturna. È poi presente una casella di testo per attribuire un nome alla scena, con un pulsante di salvataggio a fianco: tale pulsante si occupa di salvare la scena, permettendone il ricaricamento in una seconda sessione. L'utente può infine muovere il punto di vista sulla scena per esplorare meglio l'ambiente 3D, in qualsiasi momento, con i pulsanti WASD+QE+Spacebar della tastiera. L'utente può seguire un breve tutorial usufruendo del bottone "?" in alto a sinistra: si approfondisce questa funzionalità nel capitolo 6. Può infine avviare la fase di simulazione con un pulsante centrale, in alto.

Figura 4.7: Fase di costruzione della scena

4.2.1 Posizionamento e selezione

Prima di procedere alla fase successiva, ci si addentra nella realizzazione tecnica di quanto esposto fin'ora, mantenendo un linguaggio di alto livello.

In totale, le scene (intese come file con estensione .unity) sono soltanto 3: quella del menù, quella di gioco e quella dell'editor di azioni e stati. Per il menù iniziale è stato utilizzato un *canvas* (ovvero un conte-

nitore di elementi di UI) con la caratteristica di scalare in base alle dimensioni dello schermo. Al suo interno sono stati disposti bottoni caratterizzati da un `OnClickListener`, ovvero capaci di eseguire una funzione quando si verifica l'evento "click" su di essi. Per cambiare scena quando si clicca su un bottone è necessario utilizzare un metodo della libreria `UnityEngine.SceneManagement`, che permette di caricare un'altra scena. Le funzioni che implementano tale metodo sono affidate allo script `StartingSceneManager` attribuito come component al canvas.

I principali GameObject utilizzati per la fase di costruzione della scena sono invece un empty[2] chiamato *BuildingManager* ed un canvas chiamato *BuildingUI*. Il BuildingManager possiede uno ed un solo script, con il medesimo nome: può quindi essere identificato con esso. L'attributo più importante del BuildingManager è la lista di tutti i prefab che si possono posizionare in fase di costruzione, chiamata semplicemente `objects[]`. Trattandosi di un attributo pubblico, è possibile aggiungere prefab a tale lista direttamente dall'inspector. Prop, personaggi e tile[3] di terreno o altri elementi ambientali possono essere aggiunti indistintamente a questa lista: verranno poi automaticamente assegnati al menù a tendina che spetta loro, in base al loro tag. Altro attributo fondamentale è poi `pendingObject`, ovvero il GameObject che si vuole posizionare (dopo aver già premuto il bottone di aggiunta). Questo oggetto "pendente" segue la posizione del mouse ad ogni frame, come vuole la funzione di Update, e tale posizione viene allineata alle guide della griglia, se questa è attiva, grazie al metodo `RoundToNearestGrid()`. La posizione del mouse, a sua volta, viene determinata all'interno della funzione `FixedUpdate()`, più efficiente per quanto riguarda la fisica rispetto alla funzione di Update; essa agisce tramite un raycast[4] che parte dalla camera e che ricerca un GameObject con layerMask[5] "Ground", in modo da poter posizionare i prefab solo sul terreno. Il *Ground* a cui è affidata tale layerMask è un semplice piano, creato con le primitive di Unity e con la texture di una griglia in modalità "fade", dunque non invasiva, pensato per essere completato con tile di terreno che rappresentano erba, terra, piastrelle o altri tipi di terreno, che tuttavia sono opzionali: i personaggi possono camminare anche sul ground senza tile.

Il BuildingManager contiene poi i metodi `RotateObject()`, `ToggleGrid()` e `PlaceObject()`, che implementano rispettivamente la rotazione di un GameObject da posizionare, l'attivazione o disattivazione della griglia guida e il posizionamento definitivo di un GameObject. Un altro metodo, chiamato

[2]GameObject vuoto, ovvero inizialmente privo di componenti, che può essere utilizzato come supporto.

[3]Moduli quadrati utilizzati in ripetizione per costruire un terreno.

[4]Tecnica che consiste nell'inviare un raggio rettilineo a partire da un punto nello spazio 3D in una certa direzione, e rilevare gli oggetti con cui si scontra.

[5]Attributo che definisce il layer di un GameObject esclusivamente per contesti di raycasting.

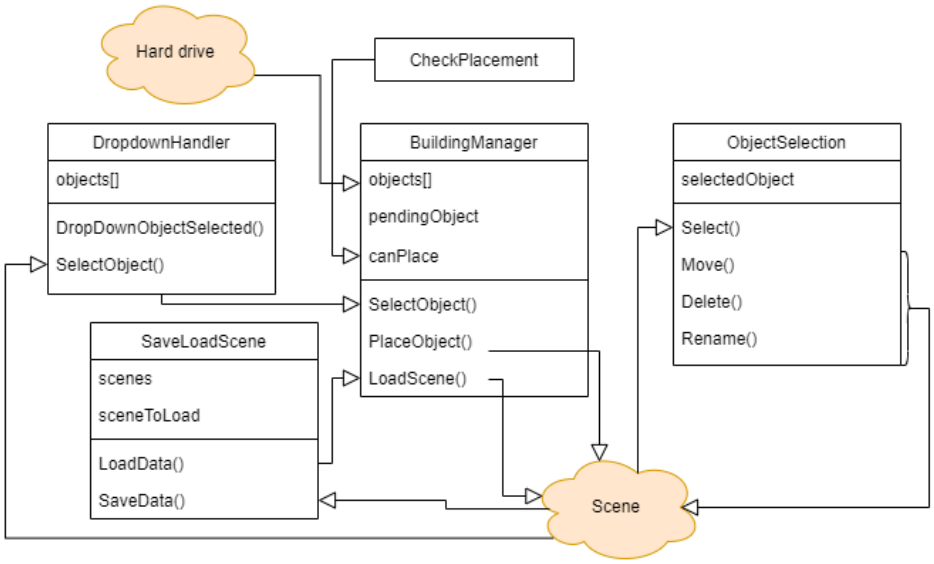


Figura 4.8: Nello schema pseudo-UML sono riportate solo le classi principali descritte, e sono evidenziati soltanto i loro attributi e metodi principali. Le frecce rappresentano richiami di altri metodi, o eventi, e non relazioni di ereditarietà.

`ToggleDay()`, cambia le impostazioni di rendering modificando il materiale dello *skybox*, ovvero dello sfondo della scena, per passare dal giorno alla notte. Il più importante è invece `SelectObject()`, il quale riceve una stringa con il nome del prefab selezionato dai menù a tendina e si occupa di cercarlo all'interno della lista `objects[]`, assegnarlo a `pendingObject`, modificarne il nome rimuovendo il suffisso automatico "(clone)" e aggiungere infine alcuni componenti al prefab per prepararlo alla simulazione: un *Rigidbody*[6] ove assente, e gli script `CheckPlacement` e `State`. Viene inoltre posto l'attributo booleano `isTrigger` del *Collider*[7] a `true`, poiché nella fase di costruzione i Collider non servono a rilevare collisioni, bensì intersezioni con altri Collider quando nuovi elementi vengono disposti al di sopra di altri. Tale attributo viene riportato a `false` in fase di simulazione.

Per quanto riguarda il componente `CheckPlacement`, si tratta di uno script dedicato a verificare eventuali intersezioni tra Collider con i metodi nativi `OnTriggerEnter(Collider other)` e `OnTriggerExit(Collider other)`: nel caso in cui ci sia un'intersezione, una variabile booleana del BuildingManager (chiamata `canPlace`) viene impostata a `false`, impedendo il posizionamento di nuovi prefab al di sopra di altri. Durante la fase di

[6]Component legato alla fisica, che definisce la massa di un GameObject ed è necessario per implementare le collisioni, insieme ad un collider.

[7]Component che definisce il volume di collisione associato ad un GameObject: tipicamente si utilizzano box collider, ovvero parallelepipedi.

simulazione, l'attributo viene riportato a `false` per ogni collider. I prefab posizionati in scena sono infine impostati come children del BuildingManager, in modo da catalogarli sotto di esso. Le relazioni tra le classi principali descritte finora sono schematizzate nella figura 4.8.

Entra in gioco poi un altro GameObject, fondamentale sia per la fase di costruzione sia per quella di simulazione, ovvero il *SelectManager*, che si occupa di gestire la selezione degli elementi già posizionati, grazie ad uno script chiamato `ObjectSelection`. La gestione della selezione degli elementi già posizionati è completamente diversa nella fase di costruzione e nella fase di simulazione: nel primo caso, l'utente vuole riposizionare i prefab in scena; nel secondo caso, l'utente vuole interagire con un elemento utilizzando il personaggio controllato per compiere un'azione su di esso. In questo paragrafo ci si limita ad illustrare il funzionamento della selezione in fase di costruzione. ObjectSelection, innanzitutto, conosce la fase di gioco grazie ad un riferimento al `SimulationManager`, altro script fondamentale (assegnato ad un empty chiamato *GameManager*) che possiede un attributo *status* pari a `0` quando ci si trova in fase di costruzione. All'interno della funzione di Update, ObjectSelection esegue un raycast per rilevare eventuali oggetti selezionati. L'attributo `selectedObject` contiene il GameObject selezionato. In caso di selezione, se ci si trova in fase di costruzione, viene mostrato il pannello contenente i bottoni di Move, Delete e Rename (come in figura 4.9), e i metodi corrispondenti a tali azioni sono implementati sempre in questo script. Il metodo `Rename()`, in particolare, modifica l'attributo `surname` del componente `State` (script) del `selectedObject`, di cui si parla nella sezione seguente. Viene infine implementato il metodo `Deselect()`, che riporta a `null` il `selectedObject`, e che può essere sempre invocato con un click destro del mouse.

La camera utilizzata in questa fase è la stessa utilizzata in fase di simulazione. Il movimento della camera per cambiare il punto di vista è delegato ad uno script chiamato `CameraManager`, impostato come componente di una *Cinemachine Virtual Camera* linkata alla *Main Camera* tramite un *Cinemachine Brain*. Si illustra questo argomento nella sezione 4.3.6.

4.2.2 User Interface e salvataggio

Per gli elementi di UI, compresi quelli già descritti, sono state utilizzate, ove possibile, le classi contenute nel package TextMeshPro. I colori scelti sono quelli della palette in figura 4.10, composta da colori analoghi, ovvero equidistanziati sul cerchio cromatico.

Il canvas BuildingUI contiene gli elementi dell'interfaccia utente sopra descritti, e ad i tre menù a tendina è assegnato uno script chiamato `Dropdown-Handler`. Questo script contiene un riferimento al BuildingManager ed il suo compito principale è quello di invocare il metodo `SelectObject()` del BuildingManager stesso, per comunicare il nome del prefab da posizionare

Figura 4.9: Il menù relativo ad un oggetto selezionato

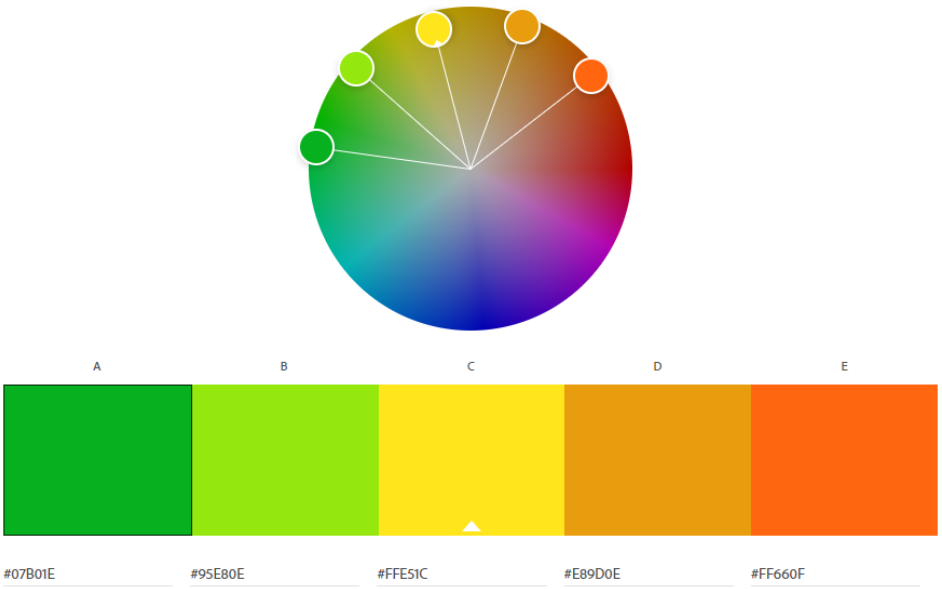


Figura 4.10: Palette utilizzata per l'UI

in scena, quando viene selezionato un elemento dai menù a tendina, a cui è assegnato un listener `onValueChanged`. Nella funzione di Start, inoltre, questo script si occupa di smistare i prefab contenuti nella lista `objects[]` del BuildingManager all'interno dei rispettivi dropdown menù in base al loro tag ("tile", "Object" o "Player"). Possiede infine un metodo, chiamato `StartSimulation()`, che si limita a nascondere tutti gli elementi dell'UI di costruzione quando si passa alla fase di simulazione.

La casella di testo che permette di assegnare un nome alla scena, come tutte le caselle di testo, è realizzata tramite la classe `TMP_InputField`. Il nome inserito viene salvato, grazie al listener `OnValueChanged` di tale campo, nell'attributo `sceneName` di `SimulationManager`. Quando si clicca sul bottone di salvataggio viene invocata la funzione `SaveData(string sceneName)` della classe SaveLoadStage di un empty chiamato *SaveManager*, che è in realtà posizionato all'interno di un'altra scena: quella del menù iniziale. A differenza di tutti gli altri GameObject, non viene distrutto durante il cambio di scena grazie al metodo `DontDestroyOnLoad()`, chiamato nella funzione di Start. Questo empty si occupa, appunto, di salvare e caricare preset di uno stage, in modo da rendere l'utente libero di non perdere una configurazione quando esce dalla simulazione. `SaveData(string sceneName)` viene dunque implementato con la classe `StreamWriter`, che permette di scrivere all'interno di un file su disco: è stato utilizzato un file in formato CSV chiamato `saved_stages.csv`. In ogni riga viene memorizzata una scena con il formato: `nome scena | nome elemento i; posizione elemento i; rotazione elemento i | nome elmento i+1 ...`, come nell'esempio in figura 4.11, dove ogni elemento i corrisponde ad uno dei GameObject posizionati in scena, tutti children di BuildingManager.

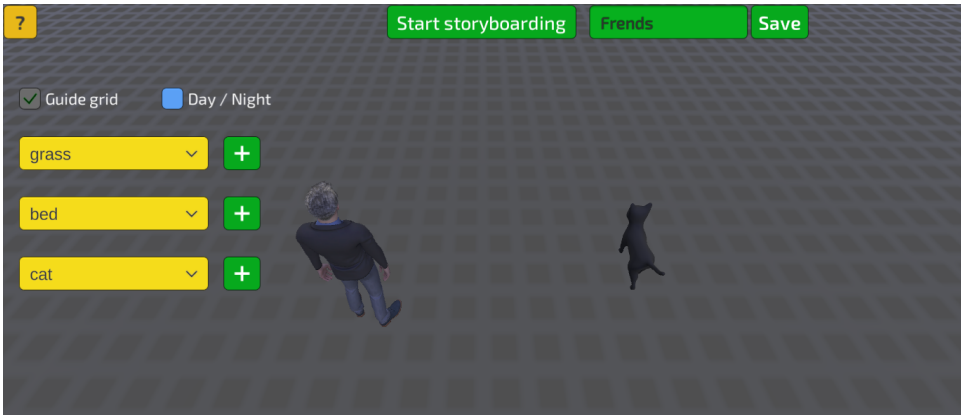


Figura 4.11: La scena, contenente due soli elementi, viene memorizzata con il nome "Friends" nel file CSV con il seguente formato: `Friends|man;-2,648798;4,926851E-17;-2,210707;359,6862|cat;-0,02818993;3,616931E-17;-1,628921;0|`

Per caricare una scena, quindi, è sufficiente ricercare in questo file, tramite la classe `StreamReader`, la riga relativa alla scena desiderata, ed istanziare ogni GameObject applicando alla sua transform la posizione e la rotazione salvate. Questo processo viene avviato dal metodo LoadData() della classe `SaveLoadStage`, ma il vero compito di istanziare i GameObject spetta al `BuildingManager`: viene infatti richiamato il suo metodo `LoadScene()`, che assegna ad ogni GameObject, inoltre, tutti i component che dovrebbero possedere se fossero stati istanziati manualmente, tramite i menù a tendina. `SaveLoadStage` si occupa, infine, di leggere il file di salvataggio per aggiungere i nomi di tutte le scene salvate come opzioni del menù a tendina che permette di selezionare la scena che si vuole ricaricare, nel menù iniziale.

4.3 Simulazione

Durante la fase di simulazione, ovvero quella di storyboarding vera e propria, l'utente può prendere il controllo dei personaggi posizionati in scena con un click del mouse sul personaggio che desidera controllare. Un testo in basso a destra indica il nome del personaggio attivo. Con i tasti WASD è possibile controllare il movimento del personaggio attivo. Il solo movimento del personaggio induce la visualizzazione dell'anteprima di frasi che indicano a quali elementi il personaggio si avvicina, verso quali è direzionato e in quali luoghi si sta muovendo. Quando il personaggio si trova abbastanza vicino ad una prop o ad un altro personaggio, poi, è possibile cliccare su di esso per veder apparire il suo nome ed una lista di azioni che è possibile eseguire, come in figura 4.12. Il click su una di queste determina vari effetti: la generazione dell'anteprima della frase che descrive l'azione; un'animazione specifica per tale azione, se presente, nel personaggio attivo ed eventualmente in quello che subisce l'azione; un'animazione generica per tale azione, in assenza di animazioni specifiche, solo nel personaggio attivo.

Figura 4.12: Azioni possibili di uno zombie relative ad un albero

Quando si clicca su un personaggio per eseguire su di esso un'azione, oltre alla lista di azioni, è visualizzabile anche un bottone che permette di prendere il controllo di tale personaggio, per renderlo il nuovo personaggio attivo. Tale bottone è visualizzabile anche se il personaggio attivo si trova troppo lontano (e non è possibile effettuare azioni).

Oltre alle azioni eseguite su altri oggetti o personaggi, esistono anche le azioni riflessive che un personaggio è in grado di eseguire da solo: per visualizzarle è sufficiente cliccare sul personaggio attivo stesso. Anche in questo caso l'azione porta alla generazione di una frase, di cui viene visualizzata l'anteprima.

Determinate azioni, in ogni caso, possono determinare un cambiamento di stato del personaggio. Il cambiamento di stato produce a sua volta l'anteprima di una nuova frase.

L'utente, in fase di simulazione, non può più intervenire sullo spostamento degli oggetti posizionati in precedenza, se non tramite personaggi attivi ed apposite azioni. Le luci del set sono gli unici GameObject che l'utente può modificare anche in fase di simulazione, per produrre cambiamenti di illuminazione diegetici. L'immagine 4.13 mostra l'interfaccia di simulazione.

Figura 4.13: Fase di simulazione

In alto al centro si trova poi la timeline, uno slider che l'utente può trascinare con il mouse per impostare il tempo corrente. Questo permette la generazione di avverbi di tempo nelle frasi. La timeline avanza automaticamente di uno step pari ad 1 secondo ad ogni cattura dell'inquadratura eseguita, ma può essere trascinato più avanti se si vuole aumentare la durata dell'inquadratura. Le frasi (e le inquadrature) verranno ordinate salvando l'informazione temporale. È possibile anche impostare da tastiera il tempo corrente, tramite la casella di testo a sinistra dello slider, nonché modificare il valore massimo dello slider (di default impostato a 10 secondi), utilizzando la casella di testo a destra: quest'ultimo valore dovrebbe corrispondere

alla durata indicativa che si prevede per la scena di cui si vuole ottenere lo storyboard.

Il pulsante in alto a destra permette di prendere il controllo della camera. Quando si controlla la camera, nessun personaggio è attivo. È possibile traslare la camera nello spazio con WASD+QE per effettuare traslazioni, e se viene tenuta premuta la barra spaziatrice i medesimi tasti eseguono le rotazioni. Si possono dunque effettuare movimenti specifici come pan[8] e tilt[9], e cambiare la lunghezza focale con uno slider. Con il tasto "P" si può eseguire, in qualsiasi momento, il salvataggio di un'inquadratura, mentre un bottone apposito produce lo storyboard in formato HTML, senza interrompere la simulazione, in modo da poterlo visualizzare anche in anteprima. Questo file viene salvato in una cartella locale ma può essere convertito in un formato migliore salvandolo come PDF dal browser. Il file HTML riporta ogni inquadratura salvata, correlata da numero crescente, descrizione modificabile, focale utilizzata e durata dell'inquadratura. Nella descrizione non sono riportare in realtà tutte le frasi visualizzate in anteprima a runtime: si effettuano alcuni tagli e riduzioni. Al fondo, si trova anche la versione prototipale dell'animatic, che si limita a mostrare le inquadrature in sequenza, con la giusta durata temporale.

Nei prossimi paragrafi si descrivono, una ad una, tutte le funzionalità presentate.

4.3.1 Controllo di personaggi

I GameObject fondamentali delegati alla fase di simulazione sono un empty chiamato *GameManager* ed un canvas chiamato *SimulationUI*, nonché il SelectManager, già citato nella sezione precedente. Il GameManager contiene alcune classi fondamentali per l'intero progetto, tra cui `SimulationManager`, e `PhraseGenerator`. In questo paragrafo ci si concentra esclusivamente su `SimulationManager`, che lavora a stretto contatto con `ObjectSelection`, componente del SelectManager. Tutte le altre classi fanno inoltre riferimento a `SimulationManager` per capire in quale fase di esecuzione ci si trova, poiché è l'unica a memorizzare tale informazione (`status = 0` in fase di costruzione, `status = 1` in fase di simulazione). Il metodo `StartSimulation()` dà effettivamente avvio alla simulazione, occupandosi di reimpostare la UI per la simulazione e porre a `false` gli attributi `isTrigger` dei collider di tutti gli elementi in scena, tramite il metodo `DeactivateTriggers()`.

Un attributo fondamentale di `SimulationManager` è `activeCharacter`, che indica il personaggio attivo. Il personaggio attivo viene impostato dal metodo `SetActiveCharacter(GameObject obj)`, chiamato proprio da `ObjectSelection` quando un personaggio viene cliccato e non ci sono personaggi attivi (come quando la simulazione è appena iniziata), oppure quando

[8] Panoramiche

[9] Rotazioni verso l'alto o verso il basso

viene cliccato il bottone per prendere il controllo di un personaggio distinto da quello corrente. Tale metodo attiva lo script `ThirdPersonMovement` e il componente `CharacterController`[10] del nuovo personaggio attivo, disattivando i componenti di quello precedente.

Il `ThirdPersonMovement` è dunque lo script assegnato ai personaggi che permette il loro movimento con i tasti WASD in fase di simulazione, quando un personaggio è attivo. Questo script contiene riferimenti all'Animator[11], al CharacterController e allo `State` del personaggio, nonché al Ground è al GameManager. All'interno della funzione di Update controlla se il personaggio sia a contatto con il suolo e si occupa di ricevere l'input da tastiera, eseguendo il Metodo `Move()` sul CharacterController ad una determinata velocità (impostata tramite una variabile pubblica); calcola inoltre la direzione verso cui il personaggio vuole puntare, eseguendo uno smoothing sulla variazione di tale angolazione tramite la variabile `turnSmoothTime`, in modo che il personaggio non cambi bruscamente direzione durante il suo movimento. Viene poi impostata a true la variabile `isWalking` dell'Animator del personaggio, la quale genera una transizione di stato di animazione verso la clip *walking*.

Il funzionamento dei componenti Animator è basato su uno schema simile ad una macchina a stati finiti: le transizioni tra un'animazione e l'altra vengono determinate dalla modifica di una variabile booleana o da un trigger, oppure dal termine di un'animazione, nel caso in cui abbia un *exit time*. Nella figura 4.14 si può osservare l'Animator di un personaggio, in cui la transizione tra l'azione di camminata e lo stato di idle[12] viene determinata da una variabile booleana, mentre le altre animazioni hanno transizioni monodirezionali, perché vengono chiamate da codice quando l'azione corrispondente viene eseguita in simulazione, e alla loro conclusione il personaggio torna allo stato di idle.

Vengono infine eseguiti, nella funzione Update, altri due metodi: `CheckNearObjects()` popola una lista con i nomi degli oggetti che il personaggio rileva vicino ad esso tramite il metodo `OverlapSphere()` della classe `Physics`: tale funzione genera una sfera centrata in un punto e con raggio a scelta, per poi rilevare i Collider che intersecano la sfera; questa lista di oggetti viene passata al componente `State` tramite il metodo `NewNearObjects(List<string> objects)`; `CheckPlace()` è invece il metodo che si occupa di capire quando il personaggio cambia luogo, dove per luoghi si intendono le aree caratterizzate da tile di terreno dello stesso tipo, o da elementi ambientali specifici, come una scala. L'attributo `place`, così come `nearObjects`, viene passato alla classe che si occupa di generare le frasi, `PhraseGenerator`, (descritta nella sezione 4.3.3) solo quando cambia

[10] Component che facilita il movimento di GameObject tramite input.

[11] Component necessario per le animazioni.

[12] Animazione tipica di un personaggio fermo, che se fosse completamente immobile risulterebbe innaturale.

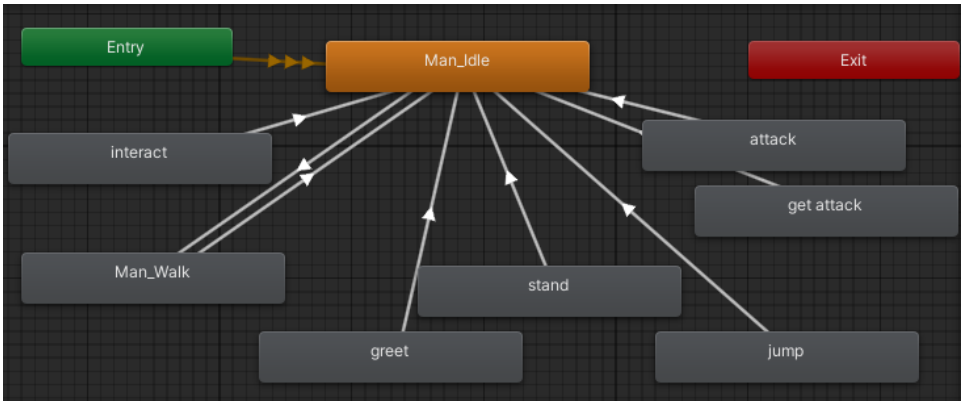


Figura 4.14: Animator di un personaggio

valore, e si vuole dunque generare una nuova frase. Lo stesso vale per il GameObject verso verso cui il personaggio attivo è direzionato, ottenuto con un raycasting e comunicato al `PhraseGenerator` tramite la funzione `UpdateForward()`, all'interno della funzione `CheckForward()`.

4.3.2 Azioni e animazioni

Quando si controlla un personaggio è possibile eseguire azioni su altri personaggi ed altre prop. La classe `ObjectSelection` si occupa, in seguito alla selezione di un elemento in fase di simulazione, anche di mostrare il menù con le azioni possibili, con un riferimento all'*ActionsPanel*, il pannello di visualizzazione delle azioni; ActionsPanel contiene il testo con il nome dell'elemento selezionato ed una *ScrollView* il cui contenuto deve essere popolato con le azioni che si vogliono visualizzare (come in figura 4.15). `ObjectSelection` preleva le azioni di interesse invocando il metodo `ReturnActions(...)` tramite la classe `ActionsDataBase`, descritta nella macro-sezione precedente, e istanzia nella ScrollView un bottone per ogni azione, il cui testo corrisponde a quello dell'azione stessa; assegna a ciascuno di questi bottoni un listener che richiama il metodo `ActionClick()` quando si clicca sul bottone. `ActionClick()` esegue alcune operazioni chiave, dal momento che rappresenta l'esecuzione di un'azione:

- notifica il componente `State` del GameObject che ha subito l'azione con il metodo `PlayAnimation(string action)` per eventuali animazioni corrispondenti all'azione subita, e con il metodo `ChangeState (string action)` per eventuali cambiamenti di stato; nulla accade nel caso in cui il GameObject che ha subito l'azione non possieda l'animazione corrispondente, o l'azione non determini un cambiamento di stato; per eseguire l'animazione via codice è sufficiente accedere al component Animator di un GameObject (se presente) e iterare le

sue `AnimationClip`, per poi eseguire `Play(string action)` se il nome dell'azione corrisponde all'animazione presente; `PlayAnimation()` controlla anche gli eventuali componenti *AudioSource* del GameObject: se si trova cioè una clip sonora con lo stesso nome dell'azione, esegue `Play()` anche su di essa;

- richiama il metodo `PlayActiveCharacterAnimation(string action)` di `SimulationManager` per l'animazione del personaggio attivo che ha eseguito l'azione; se non trova l'animazione corrispondente, questo metodo controlla ulteriormente la presenza di un'animazione chiamata "interact", pensata come predefinita, ed esegue eventualmente quella;
- invoca il metodo `GenerateSimplePhrase(...)` di `PhraseGenerator` per la generazione di una frase;
- controlla se si tratta di un'azione particolare, che non rientra in quelle generalizzabili. Per questo applicativo si è cercato infatti di generalizzare ed automatizzare gli effetti di tutte le azioni: l'unica anomala descritta finora è la camminata dei personaggi, perché avviene tramite tasti WASD anziché con il click dei bottoni. Si è voluto però implementare un paio di ulteriori azioni, *pick* e *place*, che richiedono l'imparentamento al personaggio attivo delle prop che possono essere raccolte. Quando si muove con una prop imparentata (ovvero raccolta), il personaggio attivo può continuare ad eseguire azioni su altri GameObject e sulla prop stessa, e l'imparentamento viene annullato quando esegue l'azione *place*, che posiziona la prop in un nuovo punto;
- chiama infine il metodo `HideActions()` per nascondere l'ActionsPanel, ad azione eseguita.

Figura 4.15: Lista delle azioni di "woman" su "man"

Azione "altro"

Tra i bottoni con le azioni possibili, viene aggiunto anche un bottone denominato *ohter* con una casella di testo sottostante: è possibile inserire il nome di un'azione non presente nella lista e cliccare sul bottone. Se si tratta di un'azione che determina il cambiamento di stato di un GameObject, questo viene considerato: per esempio, si supponga che un *cat* non possa eseguire l'azione *open* su una *door*; se si inserisce l'azione *open* come *other* perché si decide che in quella particolare occasione il gatto deve essere capace di aprire una porta, la porta cambia effettivamente stato, e viene aperta.

Il componente State

Il component `State` viene attribuito ad ogni GameObject posizionato in scena, in fase di costruzione, e contiene molti attributi importanti:

- `surname`, ovvero l'eventuale nuovo nome di un GameObject, attribuito dall'utente con la funzione per rinominare;
- `state`, una lista di stringhe che contiene i veri stati semantici in cui il GameObject si trova, aggiornati con il metodo `ChangeState()` usufruendo della classe `ActionsDataBase`;
- `possibleStates`, ovvero tutti i possibili stati semantici, anche quelli in cui il GameObject non si trova;
- `nearObjs`, ovvero i GameObject vicini, aggiornati grazie al `ThirdPerson-Movement`.

Feedback visivo

Per mettere in evidenza i GameObject in prossimità del personaggio, ovvero quelli su cui è possibile cliccare per veder comparire il menù delle azioni, è stato implementato un feedback visivo che consiste in un sistema particellare che genera una corona circolare in rotazione alla base del target. Contiene anche una *point light*[13] per evidenziare meglio l'oggetto in questione. Da codice, è sufficiente attivare o disattivare il sistema particellare con un riferimento al component `ParticleSystem` ed i metodi `Play()` e `Stop()`. `SimulationManager` si occupa di implementare, a tal proposito, i metodi `CreateParticle(Vector3 position)` e `DestroyParticles()`. Lo svantaggio è che il sistema particellare viene catturato nello schermo quando si effettua uno screenshot, andando ad inserire nello storyboard finale un elemento extradiegetico: si è dunque cercato di non renderlo visivamente troppo invasivo.

[13] Sorgente luminosa in cui tutti i raggi divergono da un punto centrale.

Gestione del tempo

La gestione del tempo è delegata ad un GameObject dell'interfaccia di simulazione chiamato *TimeManager*, contenente sostanzialmente uno slider e due caselle di testo per cambiare via tastiera il valore corrente ed il valore massimo dello slider. Quando lo slider cambia, è ancora la classe `SimulationManager` ad occuparsi di rilevare il cambiamento e modificare il suo attributo `time`, a cui altre classi fanno riferimento. Implementa poi i metodi `ChangeMaxTime()`, `ChangeCurrentTime()` e `ChangeCurrentTimeManual()`, per la sua gestione. Ogni volta che si effettua una cattura dello schermo è stato arbitrariamente deciso di far scorrere di 1 secondo lo slider, ma l'utente può aumentare a piacimento questo valore. Nella versione finale è possibile scorrere solo in avanti nel tempo. In precedenza, alcuni test erano stati eseguiti permettendo all'utente di catturare un'inquadratura anche ponendo lo slider ad un tempo anteriore rispetto ad un'inquadratura già salvata, e la classe dedicata alla generazione dello storyboard si occupava dell'ordinamento temporale a posteriori, per garantire la visualizzazione finale cronologica. La soluzione è stata deprecata in quanto non garante di vincoli semantici sulla causalità degli eventi.

4.3.3 Generazione delle frasi

Tra i component del GameManager si trova lo script `PhraseGenerator`, che si occupa di generare ogni tipo di frase. L'attributo più importante di questa classe è una lista di stringhe chiamata `buffer`, in cui vengono memorizzate tutte le frasi generate in una simulazione. Nel momento in cui una frase viene generata, oltre ad essere memorizzata nel buffer, viene visualizzata come anteprima sullo schermo, modificando il testo del GameObject, *PhraseText*, facente parte della SimulationUI. Il sistema supporta la generazione di frasi di vari tipi, come si può osservare nella figura 4.16:

- Azioni di un soggetto verso un complemento: generate dal metodo `GenerateSimplePhrase(string subjectSurname, string subjectName, string action, string complementSurname, string complementName, bool self)`, chiamato dal metodo `ActionClick` della classe `ObjectSelection` quando si clicca sul bottone di un'azione. Esso mette in sequenza soggetto, predicato e complemento, con alcune accortezze:
  - se i GameObject che costituiscono il soggetto e/o il complemento sono stati rinominati dall'utente, ovvero possiedono un attributo `surname` valido, viene rimosso l'articolo determinativo poiché si tratta presumibilmente di un nome proprio (e.g. *The man feeds the cat* vs. *Mario feeds the cat*);

  - i verbi vengono coniugati alla terza persona singolare con l'aggiunta della *s*, e se sono verbi frasali, ovvero seguiti da una preposizione o da un avverbio, si aggiunge la *s* soltanto al primo frammento (e.g. *Mario takes out the trash*);
  - se la variabile `self` è posta a `true` significa che il soggetto sta eseguendo un'azione da solo, poiché l'utente ha cliccato sul personaggio attivo stesso. La frase viene generata quindi senza complemento (e.g. *The man jumps*);
  - se due soggetti hanno eseguito la medesima azione sul medesimo complemento e tali azioni risultano cronologicamente adiacenti nel buffer, vengono sostituite da una sola frase in cui i soggetti vengono riuniti (e.g. *The man and the woman feed the cat*).
  - vengono generati avverbi temporali sulla contemporaneità o la sequenzialità di azioni, grazie al metodo `GenerateTimeAdverb()`, descritto in seguito.

- cambiamenti di stato: le frasi vengono generate dal metodo `GenerateStatusPhrase(string subject, string status)` chiamato dalla classe `State` quando un GameObject va incontro ad un cambiamento di stato. Su schermo, la frase viene stampata in coda all'azione che ha generato il cambiamento di stato (e.g. *The man opens the door. The door is now open*); si è però deciso di non inserire questa frase nel buffer, ovvero di non conservarla per lo storyboard in output, in quanto poco rilevante;

- spostamento dei personaggi: le frasi vengono generate dal metodo `GenerateMovementPhrase(string subject, string place)` chiamato dalla classe `ThirdPersonMovement` in occasioni distinte, ovvero ogni volta in cui un personaggio attivo inizia a muoversi, ogni volta in cui, mentre si muove, cambia il `place` in cui si sta spostando ed ogni volta in cui cambia il target verso cui il personaggio si sta dirigendo. La frase contiene informazioni sia sul luogo in cui il personaggio sta camminando, sia sul GameObject verso cui è direzionato (e.g. *The man walks on the grass, towards the tree*); anche queste frasi, in realtà, non vengono inserite nello storyboard in output, eccetto per alcuni casi particolari, ovvero quando non sono seguite da ulteriori azioni bensì da un cambiamento di personaggio attivo o da un salvataggio della vignetta. Si occupa di questa casistica il metodo `GenerateConditionPhrase()`.

- elementi in prossimità: le frasi relative all'aggiornamento degli elementi vicini al personaggio attivo vengono generate dal metodo `GenerateNearObjectsPhrase(string name, string surname, List<string> nears)`, chiamato dal componente `State` quando l'attributo `nearObjs`

cambia (e.g. *The man is near the cat and the sink*); vengono visualizzate nello storyboard finale solo nel caso analogo a quello descritto al punto precedente; se il soggetto non si trova più vicino ad alcun GameObject, veniva generata una frase apposita (e.g. *The man walks away*), ma tale funzione è stata deprecata in quanto produce informazioni irrilevanti ed eccessive;

- avverbi temporali: non sono frasi vere e proprie, ma frammenti che vengono posti prima di una frase principale, con un significato di simultaneità quando il tempo della simulazione, ritornato dal metodo `GetTime()` di `SimulationManager`, non è cambiato rispetto ad un riferimento memorizzato di tale valore temporale, e con un significato di sequenzialità se il tempo è trascorso.

Tutte le frasi vengono memorizzate nel buffer come stringhe con un prefisso numerico che rappresenta un timestamp relativo al `time` in cui è stata generata la frase.

(a) Frase di movimento

(b) Azione e cambiamento di stato

(c) Prossimità ad altri oggetti

Figura 4.16: Esempi di generazione di frasi

4.3.4 Sinonimi e WordNet

In alcune occasioni, per la generazione delle frasi, sono state utilizzate delle liste di pochi sinonimi da cui i metodi prelevano un valore random. Questo è stato implementato solo per le espressioni che, in una frase dello stesso tipo, sarebbero sempre le medesime: l'atto di un personaggio attivo di avvicinarsi a nuovi GameObject, ad esempio, viene descritto con un'espressione scelta a random da una lista che contiene espressioni come *In the meanwhile, At the same time, In the while, Simultaneously.*

Per implementare un sistema che generi variazioni linguistiche anche nei sostantivi (soggetti e complementi), nei verbi (azioni) e negli aggettivi (ad esempio i predicati nominali degli stati in cui si trova un GameObject), si è scelto di ricorrere al database semantico di WordNet. Questa funzionalità si può comodamente attivare o disattivare tramite il componente

`WordnetManager` (script) del GameManager. Tale classe si occupa di istanziare un oggetto `WordNetEngine`, grazie alla già citata libreria `Syn.WordNet`, e contiene il metodo `GetSyn(string word, string wordType)`, che restituisce un sinonimo a caso di `word`, curandosi di includere solo i synset in cui l'attributo `PartOfSpeech` (*Noun, Verb, Adjective, Adverb...*) è uguale a `wordType`. Questo metodo viene chiamato da `PhraseGenerator`, anche se non in ogni occasione possibile, ma solo per generare qualche variazione in alcune frasi.

4.3.5 Dialoghi e input vocale

I dialoghi rientrano sempre nell'ambito della generazione delle descrizioni dello storyboard, ma meritano un paragrafo dedicato. Un altro GameObject child di SimulationUI contiene l'UI per i dialoghi, che viene attivata quando l'azione eseguita da un personaggio corrisponde a "talk" o "talk to". L'UI contiene una casella di testo per inserire manualmente il discorso diretto, un toggle per attivare l'input tramite microfono, un bottone per confermare e un bottone per annullare l'operazione. In output viene mantenuta la frase che introduce il dialogo, seguita dal discorso diretto tra virgolette basse (e.g. *Mario talks to the woman. ≪How are you doing?≫*).

Figura 4.17: Interfaccia di dialogo tra personaggi in cui il discorso diretto deve essere ancora inserito

`PhraseGenerator` si occupa di generare il dialogo, tramite il metodo `GenerateDialogue()`, con un riferimento ai GameObject dell'UI necessari. Possiede inoltre i metodi `ToggleSpeech()`, `StartSpeech()` e `EndSpeech()`: il primo richiama gli altri due quando si cambia il valore del toggle che permette di inserire l'input tramite microfono, gli altri due invocano i metodi di inizio e di fine dettatura di un oggetto di classe `DictationEngine`.

Questa classe è stata creata per gestire l'input da microfono, ed utilizza la libreria `UnityEngine.Windows.Speech`, che permette di istanziare un

oggetto di tipo `DictationRecognizer` [54], per gestire la dettatura da microfono. All'interno dei metodi `StartDictationEngine()` e `CloseDictationEngine()`, l'oggetto iscrive (o disiscrive) i suoi metodi ad una serie di eventi:

- `DictationHypothesis`: invocato in modo continuo mentre l'utente sta parlando, richiama un metodo che aggiorna la frase visualizzata realtime, generando anche una variabile che esprime il livello di confidenza;
- `DictationError`: invocato in caso di errore;
- `DictationComplete`: invocato quando il sistema si arresta per qualche motivo. Nel caso in cui la causa sia un timeout, si riesegue `StartDictationEngine()`, mentre in caso di errore si esegue soltanto `CloseDictationEngine()`; di default, il timeout vale 5 secondi se il microfono non rileva audio sin dall'inizio, 20 secondi se smette di rilevare audio;
- `DictationResult`: invocato quando l'utente smette di parlare per un determinato tempo, e si suppone abbia terminato la frase; a questo evento si iscrive un metodo che al suo interno richiama direttamente il metodo `SendPhrase()` per generare la frase. Viene selezionata la frase con il livello di confidenza più alto.

Nel metodo `CloseDictationEngine()` è infine necessario richiamare `Dispose()` per rilasciare le risorse ed evitare un costo aggiuntivo nella garbage collection[14].

4.3.6 Camera e luci

Durante la fase di costruzione l'utente può spostare la camera nello spazio con i tasti WASD+QE+Spacebar per orientarsi, mentre durante la fase di simulazione, in cui i tasti WASD servono a controllare il personaggio attivo, è necessario cliccare sull'apposito bottone in alto a destra per riprendere il controllo della camera, settando a `null` il personaggio attivo. Questo bottone invoca il metodo `ControlCamera()` di `SimulationManager`, che mostra ulteriori elementi di UI quando si controlla la camera, per esempio lo slider che permette di cambiare il FOV[15]. Porta inoltre a `true` il suo attributo `controlCamera`.

La camera predefinita del progetto è stata integrata con una *Cinemachine virtual camera*, dotata di migliori funzionalità, e le è stato attribuito lo script `CameraManager`. Nella funzione di Update, questo script si occupa di rilevare l'input dell'utente, rispettando varie condizioni: mentre si digita l'input in una casella di testo, ad esempio, non deve poter essere spostata la camera per sbaglio, utilizzando una lettera che concorre al suo movimento.

[14] Processo di liberazione delle sezioni di memoria da risorse inutilizzate.
[15] Field Of View, ovvero l'angolo solido che una camera è capace di inquadrare.

La camera viene traslata nello spazio muovendosi a destra o sinistra con i tasti A e D, in avanti o indietro con i tasti Q ed E, in alto e in basso con i tasti W ed S, sempre secondo il sistema di riferimento locale. La barra spaziatrice, tenuta premuta, permette le rotazioni sui tre assi utilizzando i medesimi tasti: per la sola rotazione sull'asse y sono state utilizzate le coordinate globali anziché locali, in quanto più intuitive per questo specifico movimento. Una soluzione deprecata consisteva in una meccanica in cui la camera si muoveva solo con i tasti WASD e seguiva la posizione del mouse.

La funzione `ChangeFOV()` viene richiamata quando si cambia il valore dello slider per il FOV, e modifica direttamente l'attributo `fieldOfView` della camera. Sull'UI e sullo storyboard viene però mostrato un altro parametro, ovvero la lunghezza focale, che è inversamente proporzionale al FOV secondo la relazione:

$$FOV = 2 \cdot arctan(\frac{x}{2f})$$

dove f è la lunghezza focale, mentre x è la diagonale dello schermo [55]. Gli obiettivi delle camere, tipicamente, hanno valori standard di focali che vanno da circa 14 mm (grandangoli) a 135 mm (teleobiettivi): lo slider è stato dunque tarato su queste misure. L'effetto di obiettivi con lunghezze focali diverse si può osservare nella figura 4.18.

Figura 4.18: Differenza tra un teleobiettivo (sopra) e un grandangolo (sotto). Sulla destra si può osservare lo slider per il cambiamento della focale.

Per quanto riguarda le luci, è stato implementato un prefab costituito da una point light imparentata al modello di una lampada a fluorescenza da set cinematografico. Il prefab fa parte della categoria dei GameObject per

la costruzione dell'ambiente, e poiché si tratta di un elemento extradiegetico i personaggi non possono interagirvi. L'utente, tuttavia, può cambiare i suoi parametri in qualsiasi momento, anche in fase di simulazione, tramite un menù speciale chiamato *LightPanel*, imparentato alla SimulationUI, che viene attivato dalla classe `ObjectSelection` quando è stato selezionato un GameObject con nome "light", come in figura 4.19.

Figura 4.19: Pannello di modifica delle luci

Nel pannello si trovano due slider, che invocano due metodi di `SimulationManager` quando cambiano valore:

- `SetLightIntensity()` cambia l'intensità della sorgente tramite l'attributo `intensity`, di default posto a `5`;
- `SetLightHue()` cambia di fatto la tinta della sorgente, tramite l'attributo `color`, ma è stato utilizzato uno stratagemma per cambiare approssimativamente, in realtà, la temperatura colore, un parametro molto più importante nel contesto cinematografico, che indica quanto una luce è "fredda" o "calda" (attraverso la scala kelvin) e che le sorgenti luminose di Unity non permettono di modificare. Una luce calda, nella pratica, si manifesta con una tinta tendente all'arancione, che in una luce fredda tende invece verso il blu. Una luce neutra è bianca (e corrisponde a circa 5000 K). Sul cerchio cromatico del modello HSV[16] questa transizione tra il caldo e il freddo non si può rappresentare come sola funzione della tinta (hue), perché includerebbe il transito attraverso la zona del verde, anziché del bianco. Occorre invece considerare anche la saturazione, avvicinandosi verso l'interno del cerchio (ovvero riducendo la saturazione, fino a 0) quando ci si avvicina al verde e poi riportandola al massimo, come in figura 4.20, quando ci si riavvicina

[16]Spazio colore basato su Hue, Saturation e Value, ovvero tinta, saturazione e luminosità.

alla tinta del blu. La saturazione va quindi espressa in funzione della tinta, ad esempio con una parabola che passa per 3 punti: gli estremi della nuova scala di temperatura ed il centro del cerchio colore. Scegliendo come estremi due punti indicativamente situati sul blu e sull'arancione e risolvendo un semplice sistema lineare, si è ottenuta la parabola $sat = 9.76 \cdot hue^2 - 6.25 \cdot hue + 1$, con un risultato accettabile, che però abbassa troppo velocemente la saturazione, dando largo spazio al bianco. È stata dunque rimappata la variabile in input (hue), prima del calcolo, secondo una funzione sigmoidea [56] capace di valorizzare i punti agli estremi di un intervallo, come in figura 4.21. Si ottiene un buon risultato con la seguente sigmoide:

$$f(x) = \frac{0.64}{(1 + e^{-1.5(10.2x - 2.9)})}$$

Questo ha permesso di realizzare una pseudo-scala di temperatura utilizzando la tinta della luce.

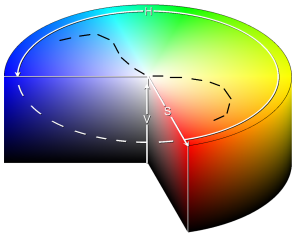


Figura 4.20: Il percorso indicativo di una scala di temperatura colore, sul modello HSV

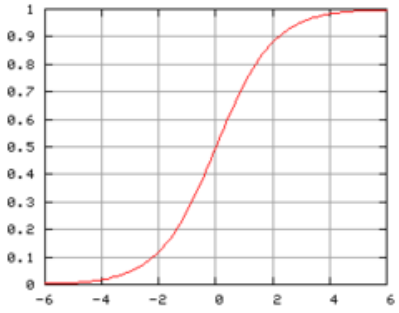


Figura 4.21: Funzione sigmoidea

Se nessuna luce viene disposta nella scena, in ogni caso, l'applicazione è perfettamente fruibile, in quanto sono presenti anche la luce ambientale e due luci direzionale di tipo *sun*, ovvero con i raggi paralleli fra di loro.

4.3.7 Generazione dello storyboard

La classe `CameraManager`, all'interno della funzione di Update, contiene un frammento di codice che si occupa di effettuare la cattura di un'inquadratura quando si preme il tasto P. L'immagine viene catturata con il metodo `CaptureScreenshot(string path)` della classe `ScreenCapture`, implementata nella librerie `UnityEngine.ScreenCaptureModule`, e viene salvata nell'indirizzo indicato con un nome così composto: `path + code + "_img" + time + ".png"`. Si sceglie cioè di memorizzare nel nome dell'inquadratura il tempo a cui è stata catturata ed un codice generato da un'altra classe, `OutputGenerator`, univoco per ogni simulazione eseguita. Si tratta di un

codice random a 6 cifre che permette di identificare tutte le immagini che devono essere utilizzate per costruire lo storyboard nel file HTML generato. Le immagini, infatti, non vengono mai cancellate dall'hard disk, se non manualmente, dunque questa catalogazione diventa necessaria. La focale utilizzata in ogni cattura, invece, viene memorizzata in una hashtable a parte.

`OutputGenerator`, componente di GameManager, contiene un solo metodo, `GenerateFile()`, che viene chiamato quando si clicca sul bottone di generazione dell'output. Il codice del file HTML viene memorizzato interamente all'interno di una stringa, con la seguente struttura:

- Dopo i tag necessari e l'header viene inserita un'intestazione con il formato `nome scena + storyboard`, seguita da un paragrafo con una descrizione testuale;
- viene inserito un titolo della prima inquadratura con il formato `"Shot # " + j`, dove `j` è un intero crescente;
- all'interno di un `<div>` di classe `flex-container`, ovvero un contenitore in cui gli elementi possono essere organizzati secondo determinate regole di stile, viene posizionata ciascuna immagine, partendo da quella con timestamp più vecchio; i timestamp vengono iterati a partire da una lista ottenuta controllando tutti i prefissi numerici del buffer delle frasi memorizzate da `PhraseGenerator`;
- nel caso in cui l'immagine non sia presente ma il timestamp sì, significa che ad un certo valore temporale i personaggi della scena hanno effettuato delle azioni, ma l'utente non ha effettuato alcuna cattura, mandando poi ulteriormente avanti il tempo, manualmente. Si tratta di fatto di un errore dell'utente, poiché ha messo in scena degli eventi senza registrarli, ma potrebbe anche verificarsi il caso in cui l'utente voglia intenzionalmente eclissarli nello storyboard, per eseguire uno stacco diegetico dopo il quale una vignetta non è più causalmente sequenziale rispetto a quella precedente. Per non perdere gli eventi accaduti, nel caso di un "errore" simile, viene comunque mostrata una vignetta corrispondente a tale timestamp, con la descrizione corretta, ma in cui l'immagine (assente) riporta un messaggio di errore;
- a fianco di ogni immagine vengono riportate la durata dell'inquadratura, semplicemente ottenuta tramite la differenza tra il timestamp corrente ed il timestamp dell'immagine precedente, e la lunghezza focale, letta dalla hashtable che memorizza la lunghezza focale utilizzata in ogni timestamp;
- al di sotto dell'immagine viene posta la descrizione, ottenuta raggruppando tutte le frasi con il medesimo timestamp, ovvero rappresentati-

ve dell'inquadratura corrente; è sufficiente assegnare `true` all'attributo `contenteditable` del paragrafo della descrizione per far sì che un utente possa comodamente effettuare modifiche al testo nel file HTML, come se si trattasse di una casella di input. Prima di generare la descrizione viene chiamato il metodo `ClearBuffer()`, che si occupa di eliminare eventuali duplicati adiacenti.

La stringa `html` viene poi scritta all'interno di un file utilizzando la classe `StreamWriter`, ed il file viene automaticamente aperto con il comando `Application.OpenURL(path)`. La figura 4.22 mostra un esempio di storyboard in output.

Figura 4.22: Due vignette di uno storyboard in output

Prototipo dell'animatic

In coda alla stringa `html`, in realtà, vengono aggiunti ulteriori elementi per realizzare la versione molto grezza di un animatic (come in figura 4.23), che si

limita a visualizzare le inquadrature dello storyboard in sequenza temporale, con il pregio di considerare le loro singole durate temporali:

- Viene inserita un'intestazione seguita da un bottone "play", che contiene un listner il quale attiva il metodo javascript `doImages()`; all'interno di tale metodo, il timing viene gestito con la funzione `setTimeout()`, che assegna tempi diversi ad ogni inquadratura in base alla loro durata; lo script javascript è inserito *inline*, proprio come il linguaggio di stile CSS;

- segue un timer, con il formato `minuti:secondi`, che fornisce un riferimento temporale per l'animatic, e che sia avvia quanto si preme "play".

L'obiettivo di questo animatic prototipale è fornire una prima visualizzazione della sequenza temporale degli eventi, per capire se il timing scelto è stato adeguato o se necessita di una revisione.

Figura 4.23: Animatic in output

4.3.8 Riassunto delle funzionalità

Il risultato finale ottenuto contiene, per riassumere tutti i moduli presentati finora, le seguenti funzionalità:

1. Permette di costruire un nuovo stage virtuale mettendo a disposizione personaggi, prop, luci ed elementi ambientali, con un sistema di trascinamento che può essere facilitato da una griglia. È anche possibile caricare uno stage già realizzato in un'altra sessione;

2. consente di avviare la simulazione nello stage creato, prendere il controllo dei personaggi inseriti e muoverli nell'ambiente, per far eseguire loro delle azioni nei confronti di altri personaggi, prop, o su se stessi. Tali azioni, selezionabili da un menù a tendina, producono animazioni corrispondenti;

3. genera descrizioni automatiche degli eventi accaduti in scena: azioni, spostamenti, cambiamenti di stato di prop e personaggi, vicinanza tra elementi, dialoghi tra personaggi, correlate da locuzioni temporali, che dipendono da una timeline;

4. permette di effettuare movimenti di camera e catturare inquadrature della scena nei momenti in cui si desidera, producendo infine uno storyboard automatico che riporta le vignette salvate, ordinate e correlate da altre informazioni, tra cui le descrizioni testuali; produce inoltre il prototipo di un animatic;

5. include un editor grafico con il quale l'utente può ridefinire a piacimento due strutture dati semantiche: quali azioni ciascun personaggio può compiere su quali prop (o personaggi); quali stati possono avere ogni personaggio ed ogni prop, quali transizioni determinano il passaggio tra questi stati e quali azioni è possibile o non è possibile effettuare in un determinato stato.

Capitolo 5

User test

5.1 Progettazione dei test

Obiettivi

Gli obiettivi degli user test si possono classificare come segue:

1. verificare l'usabilità dell'applicazione e della UI al fine di effettuare eventuali correzioni di problemi o miglioramenti delle funzionalità;

2. misurare le prestazioni dell'applicazione, in termini di tempo di esecuzione da parte dell'utente, rispetto all'ottenimento di un risultato, ed in termini di errori commessi rispetto alle indicazioni iniziali: ciò comprende la coerenza semantica delle frasi generate rispetto alla descrizione fornita nello script e la coerenza delle inquadrature, se indicate;

3. misurare le prestazioni dell'applicazione in merito alla sua effettiva utilità per la realizzazione automatica di storyboard, rispetto ad un metodo tradizionale.

Campione

Per i test è stata ottenuta la collaborazione di 15 persone: 10 studenti e studentesse di Ingegneria del Cinema e dei Mezzi di Comunicazione, 2 di Design e Comunicazione e 3 persone esterne al Politecnico di Torino. L'età delle persone è compresa tra i 20 e i 27 anni, con una media di 23,33 ed una varianza di 1,57, ed il sesso è distribuito in 60% donne e 40% uomini.

Svolgimento

I test si svolgono presso una postazione hardware con la seguente metodologia:

1. l'utente viene fatto accomodare davanti ad un computer, in cui l'applicazione è avviata sul menù iniziale, e gli si descrive brevemente il progetto, focalizzandosi sui compiti che l'applicazione permette di effettuare; si chiede conferma del fatto che l'utente sappia cosa sia uno storyboard: nel caso in cui non fosse così, viene mostrata qualche reference per spiegare brevemente il concetto (tempo indicativo: 5 minuti);

2. l'utente viene guidato nello svolgimento del tutorial, perché possa prendere familiarità con l'interfaccia e capire le funzionalità principali; si procede al punto successivo quando l'utente si sente pronto (tempo indicativo: 10 minuti);

3. all'utente viene presentato uno dei due test da svolgere, con la traccia stampata su un foglio di carta in modo che lo schermo possa essere dedicato completamente all'applicazione, e gli viene chiesto di realizzare uno storyboard corrispondente allo script presentato (tempo indicativo: 15 minuti);

4. durante lo svolgimento, si danno suggerimenti all'utente in caso di sua richiesta o di sue evidente difficoltà a continuare, e viene misurato il tempo impiegato;

5. al termine, all'utente viene sottoposto, tramite Google Form, un questionario che include un test di usabilità SUS [57] ed alcune domande libere (tempo indicativo: 10 minuti).

6. lo storyboard prodotto viene salvato come file .pdf e vengono conteggiati, solo nel test #2, gli errori commessi dall'utente, con cui si intendono frasi semanticamente errate o inquadrature incoerenti con le indicazioni fornite.

5.1.1 Tutorial

Il tutorial consiste in una serie di suggerimenti, organizzati in ordine sequenziale, forniti tramite una piccola casella di testo attivabile cliccando sul bottone "?" (in alto a sinistra in figura 5.1), sia durante la fase di costruzione sia durante quella di simulazione del sistema. Si può avanzare o tornare indietro con i suggerimenti utilizzando le frecce in basso a destra. In ogni caso, l'utente viene guidato anche a voce nell'esplorazione di tutte le funzionalità dell'applicazione.

5.1.2 Use case #1

Il test #1 è organizzato come un test relativamente "libero" ed ha come obiettivo specifico quello di misurare l'usabilità dell'applicazione in tutte

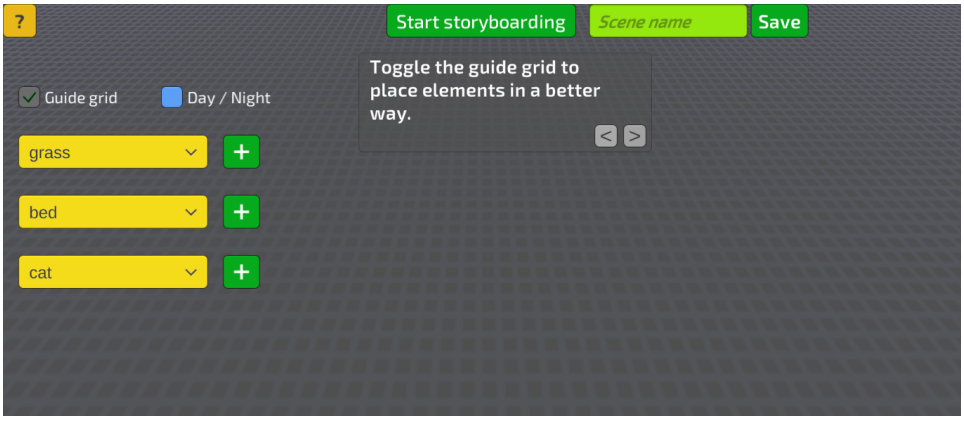


Figura 5.1: Tutorial testuale

le sue fasi, compresa quella di costruzione della scena. Viene presentata una breve traccia scritta associata alla vista in pianta della scena e dei relativi personaggi, simile ad un floor plan (figura 5.2), e si chiede all'utente di ricostruire il set virtuale e mettere in scena gli eventi descritti, senza dare indicazioni specifiche sulle inquadrature, sul numero di vignette dello storyboard o sulla loro durata. Questo caso d'uso è progettato ad hoc in modo che includa l'utilizzo di tutte le funzionalità dell'applicazione. Viene di seguito riportata la traccia fornita.

```
Jim (man) and Pam (woman) are in a bedroom with a nice
parquet, structured as shown in the map below.  Outside there
is a garden, with a zombie.
Jim picks up the plant and then moves it away from the
wardrobe.  Then he opens the wardrobe, while Pam puts some
music on the jukebox.  After that, a zombie screams from
outside while shaking a tree.  Pam leaves the room to check
and walks on the grass, towards the tree.  Then, Pam lights
the fireplace in the garden, to scary the zombie.  The zombie
hides away.  Pam goes back in the room towards Jim, but Jim
punches her, and then apologizes to her saying he was
thinking she was a zombie.
```

5.1.3 Use case #2

Il test #2 si presenta come più vincolante, rispetto al test #1: vengono infatti fornite agli utenti indicazioni chiare sulle inquadrature da realizzare e la traccia contiene, per ogni inquadratura, la descrizione esatta che si dovrebbe realizzare, al netto dei sinonimi. Il test, inoltre, non include la fase di costruzione del set: viene caricato uno stage virtuale già pronto, sia perché

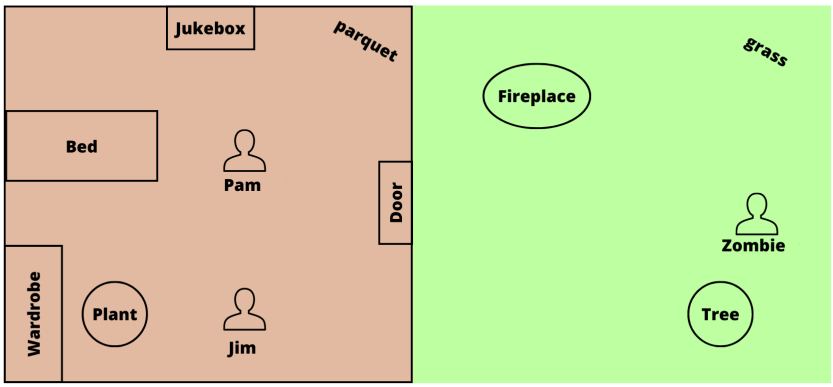


Figura 5.2: Pianta fornita con la traccia del test #1

la fase di storyboarding richiede più tempo sia perché i risultati ottenuti siano poi maggiormente confrontabili con le inquadrature target. Vengono fornite anche indicazioni circa la durata di ogni inquadratura. Segue la traccia utilizzata.

```
Total duration:  ≈ 30 sec

  1. Long shot
     Duration:  ≈ 1 sec
     Description:  The man walks on the parquet, towards the
     sink.  The man is now near the sink.

  2. Mid shot
     Duration:  ≈ 5 sec
     Description:  After that, The man washes hands in the
     sink.

  3. Mid shot
     Duration:  ≈ 5 sec
     Description:  Later, The man opens the door.  The man
     walks on the grass, towards the tree.  The man is near
     the tree and the fireplace.

  4. Close up
     Duration:  ≈ 5 sec
     Description:  Then, The cat cleans itself.
```

```
5. Mid shot
   Duration:  ≈ 15 sec
   Description:  Then, The man pets the cat.  At the same
   time, The man smiles.  Simultaneously, the cat purrs.
```

5.1.4 Questionario finale

Il questionario finale, realizzato con Google Form, è organizzato in tre sezioni: anagrafica, SUS e commenti liberi. Viene chiesto il consenso all'utilizzo dei dati (anonimi) a fini di ricerca. La sezione anagrafica si limita a richiedere l'età, il sesso ed il corso di studi.

Il questionario SUS include una serie di 10 affermazioni standard riguardanti l'usabilità del sistema: l'utente esprime, con un punteggio da 1 a 5, quanto si trova in accordo con ciascuna affermazione. I dati vengono processati, in seguito, secondo una formula che consiste nel sottrarre 1 al valore delle affermazioni dispari, sottrarre da 5 il valore delle affermazioni pari, sommare tutti i valori ottenuti e moltiplicare per 2,5 (per avere un punteggio normalizzato in centesimi). Il punteggio ottenuto è un valore nel range 0-100 da non confondere con una percentuale o un percentile: il valore 68, infatti, è il punteggio medio dei test SUS, che pone il risultato al 50simo percentile e che si prende come riferimento per distinguere un punteggio buono (sopra il 68) da uno scarso (sotto il 68). Seguono le affermazioni facenti parte del test SUS.

1. Penso che mi piacerebbe usare questo sistema frequentemente.
2. Ho trovato il sistema inutilmente complesso.
3. Penso che il sistema sia facile da usare.
4. Penso che avrei bisogno del supporto di un tecnico per poter utilizzare questo sistema.
5. Ho trovato le varie funzioni in questo sistema ben integrate.
6. Ho pensato che ci fosse troppa incoerenza in questo sistema.
7. Immagino che la maggior parte delle persone imparerebbe a utilizzare questo sistema molto rapidamente.
8. Ho trovato il sistema molto ingombrante/macchinoso da usare.
9. Mi sentivo molto sicuro usando il sistema.
10. Ho avuto bisogno di imparare molte cose prima di poter usare il sistema.

La sezione con i commenti liberi include invece le seguenti domande:

1. Quali funzionalità del sistema hai apprezzato particolarmente?
2. Quali funzionalità hai apprezzato di meno, o trovato difficili da usare?

3. Ci sono funzionalità che avresti voluto trovare nel sistema, o che suggeriresti di includere?
4. Se hai già realizzato uno storyboard con un metodo tradizionale, paragona tale metodo a quello utilizzato oggi.

Per motivi di accessibilità il questionario è posto in inglese, con la possibilità di rispondere anche in italiano.

5.2 Analisi dei risultati

Tutti i 15 tester hanno affrontato il tutorial ed il test #1. Solo 12 di loro sono poi stati sottoposti anche al test #2.

5.2.1 Dati rilevati durante il test

La tabella 5.1 riporta la durata media dei due test e la deviazione standard, visualizzata sia con il formato mm:ss sia con il formato numerico.

	Test #1	Test #2
Media	16:24	7:10
Dev.st.	3:26	2:28
Dev.st. (numerico)	0,14	0,10

Tabella 5.1: Dati sugli user test

Il tutorial si è rivelato efficace per permettere agli utenti di prendere dimestichezza con il sistema: durante i test successivi sono stati dati agli utenti consigli e suggerimenti, all'occasione, ma mai spiegazioni concettuali importanti. La durata media dei due test è sensata rispetto alle richieste delle tracce scritte e rientra nell'ordine di quella prevista. Per il test #1 circa la metà del tempo è stata impiegata dagli utenti per la ricostruzione dell'ambiente, motivo per cui il test #2, che saltava questa fase partendo da una scena già costruita, ha una durata media inferiore.

Nel test #2, in media, ogni utente ha commesso 1,50 errori, con una varianza di 0,96: in alcuni casi si tratta di vignette aggiuntive, azioni che non erano richieste dalla traccia ed errori sulla simultaneità di più azioni. Gli errori di durata delle inquadrature, invece, sono stati conteggiati solo quando l'utente sì è esplicitamente dimenticato di far avanzare la timeline, senza controllare i valori esatti, anche perché la durata suggerita nella traccia è un valore approssimativo e perché si è rivelato difficile far avanzare la timeline esattamente del valore desiderato, in quanto non permette di tornare indietro. Il grafico in figura 5.3 mostra il numero di errori per ogni utente. In alcuni casi gli utenti hanno commesso "volontariamente" alcuni errori, per

sperimentare con il sistema senza curarsi della traccia da rispettare, dunque i risultati ottenuti non sono da considerarsi troppo indicativi.

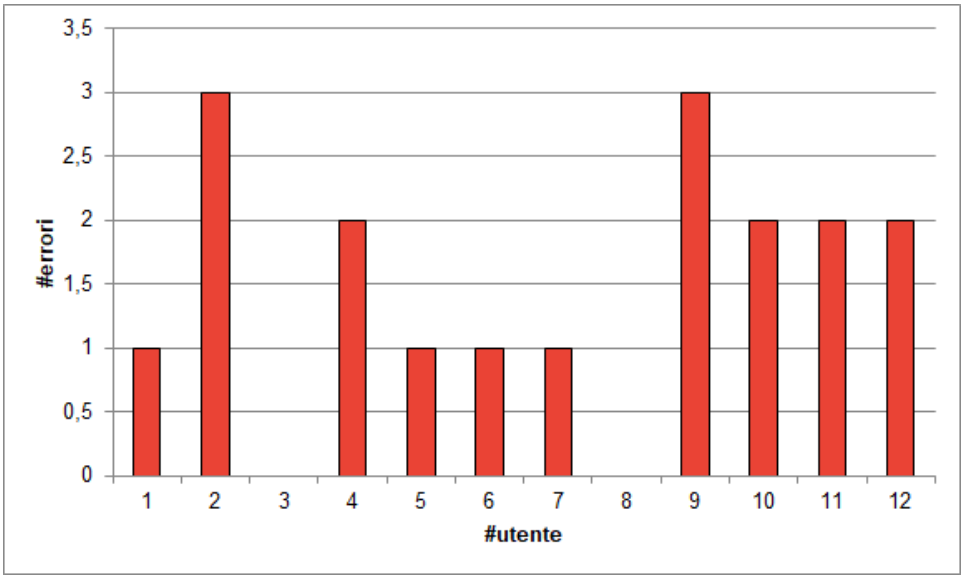


Figura 5.3: Errori commessi dai 12 utenti che hanno affrontato il test #2

Le principali difficoltà riscontrate dagli utenti durante il test sono legate ai movimenti della camera e al selezionamento dei target corretti: quando un GameObject si trova di fronte ad un altro, infatti, può essere molto difficile cliccare su quello che si trova dietro, anche se parzialmente visibile, in quanto il collider del GameObject di fronte è spesso più esteso della mesh renderizzata. In altri casi gli utenti hanno compiuto errori semantici sulla struttura logica delle azioni da far compiere ai personaggi, soprattutto quando la traccia richiede una serie di cambi tra personaggi ed azioni vicendevoli particolarmente macchinose. Altre volte è stato interpretato il bottone "other" riferito alle azioni in modo diverso da come è stato ideato: gli utenti cliccavano direttamente sul bottone, senza prima inserire l'azione nel campo di testo sottostante. Alcune difficoltà sono emerse poi con la timeline: il fatto che non si possa riportare indietro rende difficile impostare esattamente il tempo di cui si desidera avanzare, ed è poco intuitivo il fatto che il tempo vada fatto avanzare manualmente di una certa durata prima di effettuare le azioni a cui si vuole assegnare tale durata. I risultati del test #1 mostrano storyboard con pochi cambi di inquadratura, in quanto non richiesti dal testo: gli utenti hanno preferito inquadrare l'intero ambiente per tutta la durata della simulazione, in modo da non dover pensare anche alla gestione della camera. La gestione di questo elemento ha suscitato perplessità in vari casi, e in rari casi una certa confidenza, forse dovuta a soluzioni simili già sperimentate. Il cambio di elevazione è stato corretto

dopo i test assegnandolo ai tasti Q ed E anziché ai tasti W ed S, in quanto analogo a molti videogiochi, come osservato da alcuni utenti.

5.2.2 SUS e osservazioni

Il punteggi calcolati con il SUS sono riportati nella figura 5.4, con una media di 78,16 e una deviazione standard di 8,03, rientrando nella catalogazione di "buona usabilità".

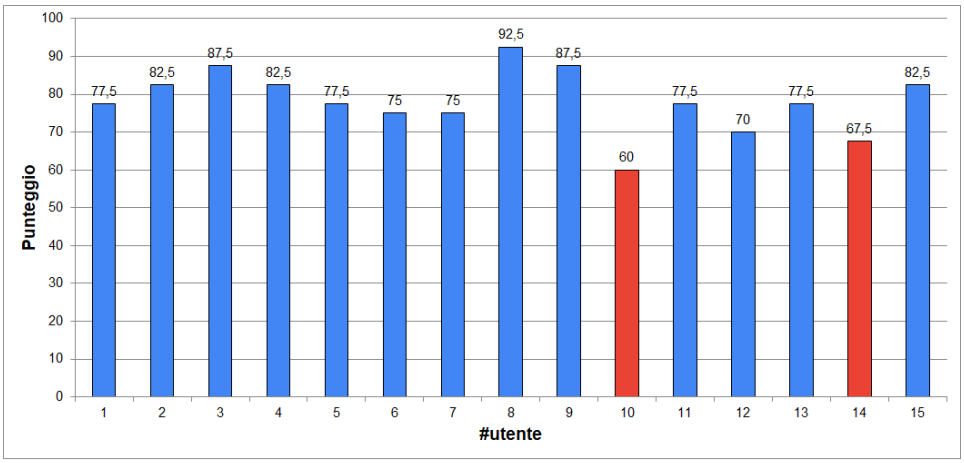


Figura 5.4: Punteggi ottenuti da ciascun utente con il SUS

Due punteggi si trovano al di sotto della soglia di 68, rientrando nel range "marginale" tra l'accettabile e il non accettabile: il più basso di questi punteggi, tuttavia, è stato ottenuto dall'unico utente a cui è stato fatto svolgere il test #2 prima del test #1, a fini di sperimentazione. Il test #1, essendo meno vincolante, permette agli utenti di sperimentare meglio con il sistema, e soprattutto non contiene troppe richieste come il test successivo, legate al tempo e alla camera. L'overflow di requisiti ha portato l'utente ad un'esperienza di usabilità meno piacevole.

Le affermazioni dispari del SUS indicano buona usabilità, mentre quelle pari cattiva. Dal grafico in figura 5.5 si possono osservare i punteggi medi assegnati a ciascuna affermazione. Si rilevano gli indicatori di usabilità più scarsa, assegnati alle affermazioni 1 e 8, riferite a quanto spesso un utente utilizzerebbe il sistema e alla macchinosità dei processi. I punti di forza stanno invece nelle affermazioni 6 e 7, riferite all'inconsistenza del sistema (bassa) e alla rapidità con cui si suppone che gli utenti possano imparare ad utilizzarlo.

I commenti liberi, infine, hanno riportato interessanti osservazioni. Gli utenti hanno apprezzato la facilità e la velocità con cui si può costruire un set virtuale, la versatilità del sistema nel permettere ai personaggi di eseguire azioni e animazioni, la generazione delle descrizioni ed il menù di

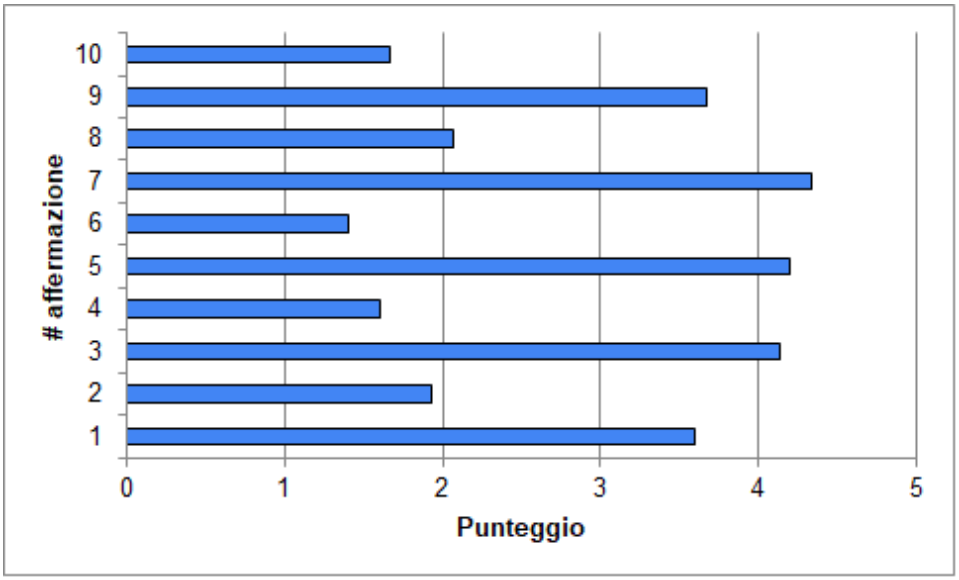


Figura 5.5: Media dei punteggi di 1 a 5 assegnati dagli utenti ad ogni affermazione del SUS

personalizzazione delle azioni e degli stati, anche se non faceva parte dei test: è stato infatti mostrato ad alcuni utenti interessati. In alcuni casi è stato molto apprezzato l'animatic, per quanto semplice, in quanto effettivamente utile per uno studio del timing.

I movimenti di camera rientrano invece nelle criticità evidenziate: molti utenti non trovano intuitivo l'utilizzo dei tasti WASD, un utente suggerisce di utilizzare il tasto destro del mouse per le rotazioni, anziché la barra spaziatrice combinata con i medesimi tasti. Un altro utente suggerisce di inserire un riquadro interno alla camera, in modo da visualizzare meglio l'inquadratura che si sta per catturare, anziché catturare tutto ciò che viene renderizzato.

Relativamente alla domanda 3, relativa alle feature che gli utenti aggiungerebbero al sistema, si menzionano la necessità di più shortcut (inclusa una per annullare l'ultima azione), la necessità di modificare il timing anche nell'output finale e di tornare indietro nel tempo durante la simulazione, ed una miglior gestione dell'azione "other". Un utente suggerisce di suddividere in due step paralleli la fase di storyboarding: nel primo ci si occupa di tutti gli eventi che devono accadere in scena, e solo in seguito si impostano le inquadrature. Utenti più esperti hanno poi sentito la necessità di scegliere più velocemente l'ottica da un set predefinito, anziché scorrere con lo slider tra tutti i valori del range.

Sul paragone finale che è stato chiesto agli utenti tra il metodo di storyboarding utilizzato ed un metodo tradizionale, infine, sono state evidenziate

alcune virtù del sistema: permette di costruire rapidamente un ambiente senza doverlo disegnare, tiene conto degli spazi 3D e delle distanze, permette di salvare un ambiente e riutilizzarlo in momenti diversi. D'altro canto permette meno libertà a causa del set limitato di modelli ed animazioni che fornisce.

5.2.3 Esempio di output

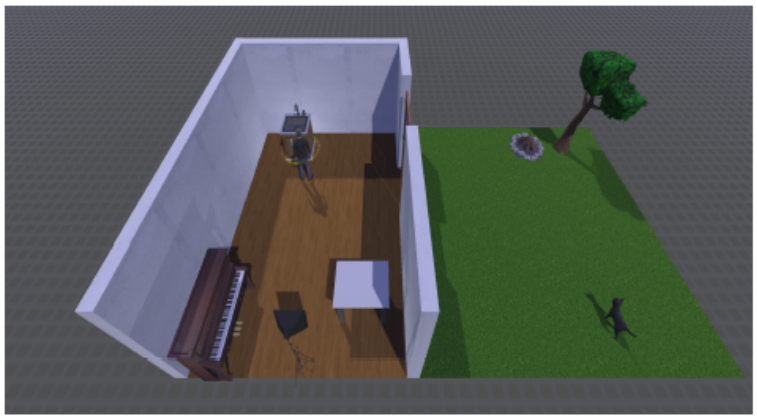
Si conclude con un esempio di storyboard prodotto per il test #2.

Test#2 Storyboard

Panels descriptions are editable! Click on them to make adjustments.

Go to 'print' and select 'save as pdf' to download your storyboard!

Shot #1



Duration:
1 sec

Focal
length:
60 mm

The man walks on the parquet, towards the sink. The man has moved to the sink.

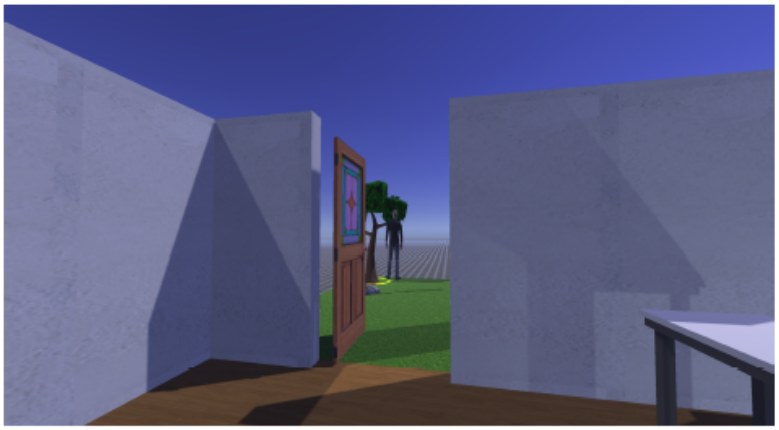
Shot #2

Duration:
6 sec

Focal
length:
60 mm

After that, the man washs hands in the sink.

Shot #3

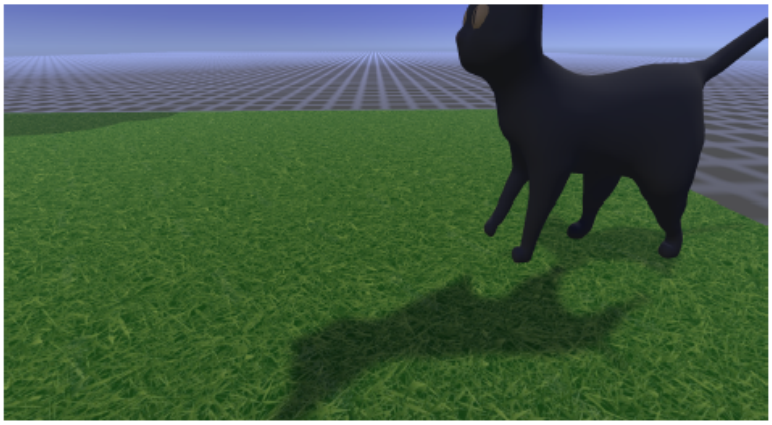


Duration:
6 sec

Focal
length:
60 mm

Later, the man opens the door. The man walks on the grass, towards the tree. The man is now near the tree and the fireplace.

Shot #4

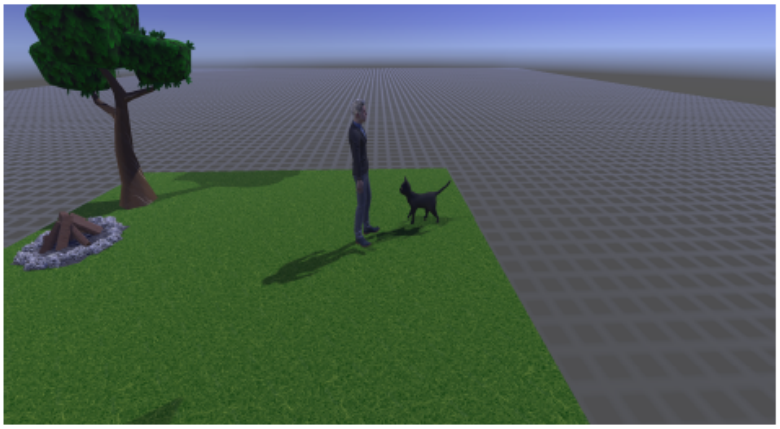


Duration:
6 sec

Focal
length:
60 mm

Later, the cat cleans itself .

Shot #5



Duration:
16 sec

Focal
length:
60 mm

Following, the man smiles. In the while, the man pets the cat. In the while, the cat purrs.

Capitolo 6

Conclusioni

Il progetto descritto in questa tesi ha affrontato problemi appartenenti a campi molto distinti. Primo fra tutti, il problema semantico, strettamente connesso alla funzionalità di generazione automatica delle descrizioni. È stata realizzata un'interfaccia che permette, non al programmatore bensì ad uno sceneggiatore o ad un utente finale di qualunque tipo, di ridefinire il dizionario delle azioni, gli stati dei personaggi e i rapporti di causa-effetto che determinano i cambiamenti di stato. Su questo punto, molti compromessi sono stati decisi, per forza di cose, creando un modello semplificato della realtà, ma logico e funzionante. L'altra sfida principale consisteva nel generalizzare lo strumento, permettendo all'utente di utilizzare i suoi modelli, con le loro animazioni, integrandoli velocemente nel sistema, e di automatizzare l'esecuzione delle animazioni quando vengono scelte tramite la UI. La generalizzazione include anche la realizzazione dello stage virtuale. Sono poi state sperimentate altre funzionalità come la gestione del tempo, l'input vocale, la generazione di sinonimi, lo studio di vari contesti semantici al fine di generare frasi di diverso tipo.

6.1 Sviluppi futuri

Il sistema presenta dunque innumerevoli possibilità di ulteriori sviluppi, che possono tuttavia orientarsi in direzioni ben distinte, in base agli obiettivi che si vogliono raggiungere e a quali funzionalità si vogliono potenziare, a discapito di altre.

Semantica

Nell'ambito semantico, la generazione delle frasi può essere potenzialmente migliorata quanto si vuole, rendendo le descrizioni più simili possibili al linguaggio naturale, tramite l'aggiunta di eventi sempre più complessi. Un rischio sempre presente, tuttavia, è quello di programmare eventi sempre

più specifici, allontanandosi dal concetto di generalizzazione dell'applicazione, che ne garantisce versatilità e usabilità. La generazione della descrizione "perfetta", dunque, potrebbe richiedere un approccio del tutto diverso nell'ambito del Natural Language Processing.

La gestione degli stati dei personaggi e delle prop, d'altro canto, potrebbe facilmente trovare il suo naturale sviluppo in un approccio più complesso, che contempli, ad esempio, più reazioni possibili in seguito ad un'azione, dunque più stati distinti, e anche la possibilità che un'azione non venga eseguita con successo, ma determini un effetto secondario. Questo implica che l'utente possa compiere un'ulteriore decisione, sempre tramite la UI, quindi sussiste il rischio di trasformare ogni micro-evento in un'azione troppo macchinosa, nel caso in cui la UI non sia progettata bene, e sia invece troppo invadente.

Tempo

Lo spostamento dei personaggi consiste in un'azione particolare, effettuata da tastiera tramite i tasti WASD, che genera frasi specifiche: questo tipo di interazione potrebbe essere sostituito da un algoritmo di pathfinding[1] che, data la posizione iniziale e quella finale del personaggio, trovi in automatico il percorso e lo metta in atto, evitando gli ostacoli. Questo tipo di approccio permetterebbe di conservare informazioni esatte su tutti gli eventi accaduti in ogni istante di tempo, e di poter eventualmente riportare indietro la timeline, prestando cura a gestire anche il ripristino di stati precedenti degli elementi.

Costruzione del set

Maggiore libertà all'utente potrebbe essere data durante la fase di costruzione permettendogli di imparentare elementi, e di conseguenza posizionarli uno sopra l'altro, oppure uno dentro l'altro, aumentando però la complessità semantica dell'interpretazione testuale degli eventi. Questa funzionalità è stata testata nel caso specifico del "pick and place", ma l'approccio per una sua generalizzazione deve essere diverso. Altre feature integrabili riguardano la costruzione del terreno: si potrebbe facilitare il posizionamento dei tile con uno strumento più rapido o chiedendone le dimensioni in input, oppure utilizzando direttamente i *terrain* di Unity [58], che permettono di modellare un terreno ed includono strumenti di sculpting, painting ed altro: si tratta però di uno strumento dell'editor, utilizzato dai level designer, e andrebbe pertanto convertito in una versione semplificata presente nella build.

[1] Algoritmo con cui un'intelligenza artificiale determina il percorso più breve per giungere ad un traguardo evitando ostacoli insormontabili.

Storyboard e animatic

Ulteriori informazioni che potrebbero essere utili nello storyboard in output riguardano i movimenti di camera e dei personaggi, rappresentati tradizionalmente con delle frecce e, nel caso dei movimenti di camera, spesso riportati testualmente secondo una specifica terminologia cinematografica. L'animatic, inoltre, si potrebbe generare come girato vero e proprio, anziché come semplice sequenza delle vignette dello storyboard, tenendo traccia delle azioni compiute in ogni istante da ogni personaggio e riproducendole al momento della visualizzazione dell'animatic. Esiste già, tuttavia, un recente package di Unity chiamato Timeline [59], che fornisce una finestra in grado di realizzare compiti molto simili, permettendo di affiancare animazioni su una linea del tempo; è pensato principalmente per realizzare le cutscene di un videogioco.

Suoni

L'aspetto audio è stato solo superficialmente toccato all'interno di questo progetto: infatti, il sistema riproduce suoni - ove presenti - corrispondenti al nome delle azioni effettuate o dei nuovi stati; tuttavia si tratta di informazioni che vengono perse nell'output finale, e che potrebbero invece essere molto utili nell'ottica della realizzazione di una *scratch track*, ovvero della soundtrack preliminare, spesso sviluppata di pari passo con l'animatic.

Build per il web

Utilizzando una libreria grafica per il web, potenzialmente, si potrebbe realizzare la versione web di questa applicazione per avere una serie di vantaggi:

- si potrebbero pianificare user test su larga scala, che permettano di avere risultati molto più significativi, risolvere bug e raccogliere eventuali dati sulle esigenze degli utenti che lavorano o studiano in un ambito in cui questo sistema può risultare veramente utile;
- si potrebbe fornire facilmente un servizio, eventualmente con funzionalità premium;
- permettendo agli utenti di condividere i loro modelli e di arricchire i database semantici, il sistema si migliorerebbe da solo ogni giorno, estendendo la sua libreria di modelli e perfezionando i dizionari delle azioni e degli stati, in un'ottica di co-creazione.

Bibliografia

[1] Okun J. A., Zwerman S., *The VES handbook of Visual Effects*, cap 2 (Elsevier Inc, 2010)

[2] Glebas F., *Directing the Story: Professional Storytelling and Storyboarding Techniques for Live Action and Animation* (Focal Press, 2009)

[3] *How to Storyboard: A Basic Guide for Aspiring Artists*, https://design.tutsplus.com/articles/how-to-storyboard-basic-guides-for-aspiring-artists--\cms-30962

[4] Truong K. N., Abowd G. D., Hayes G. R., *Storyboarding: An Empirical Determination of Best Practices and Effective Guidelines* (Conference on Designing Interactive Systems, 2006)

[5] Wikipedia, *Storyboard*, https://it.wikipedia.org/wiki/Storyboard

[6] Canva.com, *Modelli per storyboard*, https://www.canva.com/it_it/storyboard/modelli/

[7] Milanote.com, *The online storyboard creator*, https://milanote.com/product/storyboarding

[8] Sherman A., *StoryboardThat*, https://www.storyboardthat.com/it/

[9] Studio Binder, *The Ultimate Online Storyboard Creator*, https://www.studiobinder.com/storyboard-creator/

[10] PanelForge, *The ultimate visual storytelling software*, panel-forge.com

[11] Chen S., Fan L., Chen C., Su T., Li W., Liu Y., Xu L., *StoryDroid: Automated Generation of Storyboard for Android Apps* (IEEE/ACM 41st International Conference on Software Engineering, 2019)

[12] Baldwin T., Ye P., *Towards Automatic Animated Storyboarding* (Twenty-Third AAAI Conference on Artificial Intelligence, 2008)

[13] Epic Games, *Unreal Engine*, `https://www.unrealengine.com/en-US`

[14] Zhang Y., *Generating Animation from Screenplays* (Eighth Joint Conference on Lexical and Computational Semantics, 2019)

[15] Pizzi D., Cavazza M., Whittaker A., Lugrin J., *Automatic Generation of Game Level Solutions as Storyboards* (Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, 2008)

[16] IO Interactive, *Hitman game series*

[17] Chen S. et al., *Neural Storyboard Artist: Visualizing Stories with Coherent Image Sequences* (ACM International Conference on Multimedia, 2019)

[18] Microsoft edge blog, *Microsoft Edge now provides auto-generated image labels*, `https://blogs.windows.com/msedgedev/2022/03/17/appears-to-say-microsoft-edge-auto-generated-image-labels/`

[19] The NetHack DevTeam, *NetHack* (Software libero, 1987)

[20] Game Freak, *Pokémon Versione Diamante* (Nintendo, 2006)

[21] Angelo Licati, *Dark Resurrection: Keepers of the Force* (2007)

[22] Team Dakota, *Project Spark* (Microsoft Studios, 2014)

[23] Workman M., *Cine Tracer*, `https://store.steampowered.com/app/904960/Cine_Tracer/` (Cinematography Database, 2018)

[24] Visual Blasters LLC, *FlipaClip* (2012)

[25] Ghibaudi L., Sanna A., *Ricostruzione automatica dello storyboard mediante il controllo interattivo di personaggi 3D* (Politecnico di Torino, 2022)

[26] Wikipedia, *Automa a stati finiti*, `https://it.wikipedia.org/wiki/Automa_a_stati_finiti`

[27] Wikipedia, *Behavior tree*, `https://en.wikipedia.org/wiki/Behavior_tree_(artificial_intelligence,_robotics_and_control)`

[28] Obsidian Entertainment, *Fallout: New Vegas* (Bethesda Softworks, 2010)

[29] Arnaud Roques, *PlantUML* (GNU General Public License), https://plantuml.com/state-diagram

[30] XState, *Stately*, https://stately.ai/

[31] Roscosmos, *DRAKON editor*, https://drakon-editor.sourceforge.net/editor.html

[32] Blender Foundation, *Blender reference manual*, https://docs.blender.org/manual/en/latest/

[33] *Python documentation*, https://www.python.org/doc/

[34] Blender Foundation, *UPBGE*, https://upbge.org/#/

[35] Unity Technologies, *Unity User Manual*, https://docs.unity3d.com/Manual/index.html

[36] Unity Technologies, *Unity Asset Store*, https://assetstore.unity.com/

[37] Kaydara, Autodesk, *FBX file format*

[38] Microsoft, *C# documentation*, https://learn.microsoft.com/it-it/dotnet/csharp/

[39] Microsoft, *Visual Studio homepage*, https://visualstudio.microsoft.com/it/

[40] Unity documentation, *Prefabs*, https://docs.unity3d.com/Manual/Prefabs.html

[41] Patrick McCarthy, *GitHub NuGetForUnity repo*, https://github.com/GlitchEnzo/NuGetForUnity

[42] Miller G.A., *WordNet: A Lexical Database for English*, (Communications of the ACM Vol. 38, No. 11: 39-41., 1995)

[43] Università di Princeton, WordNet, https://wordnet.princeton.edu/

[44] REVARN Cybernetics LLP, *Syn Developer Network*, https://developer.syn.co.in/tutorial/index.html

[45] Unity Technologies, *TextMeshPro documentation*, https://docs.unity3d.com/Manual/com.unity.textmeshpro.html

[46] Unity Technologies, *Cinemachine documentation*, https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/index.html

[47] *CsvHelper*, https://joshclose.github.io/CsvHelper/

[48] Microsoft, *.NET API browser*, https://learn.microsoft.com/en-us/dotnet/api/?view=netframework-4.8&preserve-view=true

[49] Unity, *Scripting API*, https://docs.unity3d.com/ScriptReference/

[50] Unity Asset Store, *Furniture Asset Pack*, https://assetstore.unity.com/packages/3d/props/furniture/glassofcoins-furniture-asset-pack-200983

[51] Epic Games, *Sketchfab*, https://sketchfab.com/feed

[52] Adobe Incorporated, *Mixamo*, https://www.mixamo.com/

[53] *JavaScript documentation*, https://devdocs.io/javascript/

[54] Microsoft, *Input vocale in Unity*, https://learn.microsoft.com/it-it/windows/mixed-reality/develop/unity/voice-input-in-unity

[55] Wikipedia, *Focal Length*, https://en.wikipedia.org/wiki/Focal_length

[56] Wikipedia, *Funzione sigmoidea*, https://it.wikipedia.org/wiki/Funzione_sigmoidea

[57] Brooke J., *SUS - A quick and dirty usability scale* (Redhatch Consulting Ltd, 1995)

[58] Unity Technologies, *Terrain*, https://docs.unity3d.com/Manual/script-Terrain.html

[59] Unity Technologies, *Timeline*, https://docs.unity3d.com/Packages/com.unity.timeline@1.2/manual/index.html