POLITECNICO DI TORINO

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

Tesi di Laurea Magistrale

Ricostruzione automatica dello storyboard mediante il controllo interattivo di personaggi 3D

Relatore
prof. Andrea SANNA

Candidato
Lorenzo GHIBAUDI

firma del relatore

. . . . . .

firma del candidato

. . . . . .

Anno Accademico 2021-2022

Sommario

Durante la fase di pre-produzione di un prodotto, ci sono diversi step da seguire. Uno di questi è la realizzazione dello storyboard o storyboarding, che viene però spesso tralasciata in quanto definita uno spreco di risorse. Chi ne fa utilizzo, ha diversi vantaggi nelle fasi successive, a fronte di una maggiore quantità di tempo e denaro spesi.

Al giorno d'oggi, ci sono modalità di realizzazione differenti e più moderne rispetto all'iniziale "carta e matita", come ad esempio l'uso di software che permettono di disegnare oppure altri che forniscono librerie di ambienti e personaggi per comporre la propria storia.

Alcuni studi hanno avuto lo scopo di realizzare un'animazione automaticamente partendo da una sceneggiatura o da uno storyboard, ma non il processo contrario.

In questa tesi, quindi, è stato realizzato un applicativo scritto in codice Python all'interno del software Blender, in grado di realizzare in modo automatico lo storyboard. Questo avviene in tempo reale attraverso la performance dell'utente il quale muove i personaggi e la macchina da presa all'interno di una scena 3D appositamente realizzata. Il sistema interpreta ciò che l'utente fa compiere ai personaggi e l'inquadratura da lui scelta, realizzando sketch dopo sketch un intero storyboard composto da ogni immagine salvata con relativa descrizione testuale.

Questo è stato realizzato attraverso tre casi d'uso i quali, pur utilizzando tutti le stesse funzioni, mostrano come siano contemplati diversi livelli di complessità e come il sistema sia in grado di generare frasi in modo intelligente, oltre che automatico.

Ringraziamenti

Desidero ringraziare il professore Andrea Sanna per avermi dato l'opportunità di lavorare a questo progetto e per essere sempre stato disponibile in tutte le fasi di lavorazione.

Vorrei ringraziare i miei compagni di corso, divenuti ormai grandi amici, che hanno reso ogni giornata di studio più leggera e divertente.

Ringrazio tutti gli amici di una vita, quelli vicini e quelli lontani, i quali sono sempre stati presenti, e restano sempre le migliori persone con le quali condividere del tempo.

Infine, ringrazio la mia famiglia, Cinzia, Franco, Nicolò ed Edoardo. Grazie ai miei genitori per avermi supportato e sostenuto sempre, ed avermi trasmesso i valori della vita. Grazie ai miei fratelli, per essere i miei migliori amici e coloro che mi trasmettono una serenità indescrivibile.

Indice

Elenco delle tabelle VI

Elenco delle figure VII

1 Introduzione 1
- 1.1 Obiettivo della tesi . . . 2
- 1.2 Linee guida storyboard . . . 3

2 Stato dell'arte 9
- 2.1 Software per storyboard . . . 9
- 2.2 Ricerche e studi . . . 12

3 Tecnologie 17
- 3.1 Blender . . . 17
  - 3.1.1 Versione 2.9 . . . 18
  - 3.1.2 Versione 2.79 . . . 19
- 3.2 Python . . . 23
  - 3.2.1 Librerie utilizzate . . . 23
  - 3.2.2 Creare una propria libreria . . . 24

4 Funzionamento dell'applicativo 27
- 4.1 Requisiti . . . 27
- 4.2 Design . . . 28
  - 4.2.1 Personaggi . . . 28
  - 4.2.2 Gruppo di personaggi . . . 30
  - 4.2.3 Macchina da presa . . . 30
  - 4.2.4 Altri oggetti . . . 31
- 4.3 Sviluppo . . . 32
  - 4.3.1 storyboardlibrary.py . . . 32
  - 4.3.2 vocab.py . . . 35
  - 4.3.3 Personaggi (box) . . . 35

4.3.4 Personaggi (armatura) . . . 38
4.3.5 Personaggi (altre componenti) . . . 40
4.3.6 Gruppo di personaggi . . . 41
4.3.7 Macchina da presa . . . 42
4.3.8 Macchina da presa (altre componenti) . . . 44
4.3.9 Oggetto “Ground” . . . 44
4.3.10 Altri oggetti . . . 46
4.3.11 Eccezioni . . . 47
4.4 Generazione delle frasi . . . 47

5 Use cases e test 51
5.1 Use cases . . . 51
5.1.1 Use case 1 . . . 51
5.1.2 Use case 2 . . . 52
5.1.3 Use case 3 . . . 54
5.2 Test . . . 55

6 Analisi risultati 59

7 Conclusioni e sviluppi futuri 69

Appendice A 71
A.1 Creazione del file html . . . 71
A.2 Creazione e inserimento della card . . . 71
A.3 Cancellazione dell’ultima card salvata . . . 72
A.4 Questionario . . . 73

Bibliografia 77

Elenco delle tabelle

4.1 Riassunto elementi base box 38
4.2 Riassunto elementi base armatura 40
4.3 Riassunto elementi base camera 43
4.4 Riassunto elementi base camera 46

Elenco delle figure

1.1 Parte dello storyboard de “Il Re Leone” [2] 2
1.2 Storyboard del videogioco Ryzom [3] 3
1.3 Screenshots del tool 6
1.4 Screen di una scena su PlotBoard 7
2.1 Interfaccia TVP Animation 10
2.2 Interfaccia ToonBoom Storyboard 10
2.3 Storyboard creato su storyboardthat.com 11
2.4 Esempio di utilizzo dello schema basato su Spanning Tree 12
2.5 Fasi del processo che porta allo storyboard finale 13
2.6 Esempio del sistema in azione 14
2.7 Funzionamento dell’applicativo 15
2.8 Architettura del sistema: segmentazione in diversi blocchi funzionali. Le frasi vengono semplificate e utilizzate per generare l’animazione 15
3.1 Personaggi modellati nella versione 2.9 19
3.2 Ambiente di lavoro con la v2.79 con logic bricks e script Python . . 20
3.3 Evidenziata la parte comune a tutti i sensori 21
4.1 Le 3 componenti essenziali di un personaggio 28
4.2 Le componenti aggiuntive di un personaggio (camera) 29
4.3 Più elementi compongono la casa 31
4.4 Esempi di frasi relative alle azioni 48
4.5 Esempi di frasi relative alle informazioni di contesto 49
5.1 Storyboard del caso d’uso 1 realizzato con l’applicativo 52
5.2 Parte dello storyboard de “Lilo & Stitch” che ha ispirato questo caso d’uso [52] 52
5.3 Storyboard del caso d’uso 2 realizzato con l’applicativo 53
5.4 Storyboard realizzato per un contest della RedBull dal quale è stato preso spunto per questo caso d’uso [53] 54
5.5 Storyboard del caso d’uso 3 realizzato con l’applicativo 55
5.6 Come sono state inserite le correzioni 56
5.7 Le due visuali del personaggio cane 57
6.1 Grafico risultati SUS 59

6.2 Andamento per ogni domanda 60
6.3 Tempo impiegato (minuti) 61
6.4 Sketch realizzati 62
6.5 Storyboard tester 1 63
6.6 Storyboard tester 2 63
6.7 Storyboard tester 3 64
6.8 Storyboard tester 4 64
6.9 Storyboard tester 5 65
6.10 Storyboard tester 6 65
6.11 Storyboard tester 7 66
6.12 Storyboard tester 8 66
6.13 Storyboard tester 9 67
6.14 Storyboard tester 10 67
6.15 Storyboard tester 11 68
6.16 Storyboard tester 12 68

Capitolo 1

Introduzione

Lo storyboard, che dall'inglese, letteralmente, significa "tavola della storia", si potrebbe tradurre come "sceneggiatura disegnata", in quanto indica una rappresentazione grafica delle varie inquadrature che, poste in sequenza seguendo un ordine cronologico, creano l'intera storia del film, dell'animazione o del fumetto che si sta realizzando [1].

La realizzazione dello storyboard, è una delle fasi presenti nella, più grande, fase di pre-produzione del prodotto, insieme ad altre quali l'istituzione di un budget, il casting, la scelta e/o la realizzazione delle location, analizzare la sceneggiatura per dividerla in scene e stabilire tutto il necessario per ognuna, ecc.

Ma perché è utile, se non necessario, realizzare lo storyboard? Il più importante motivo è che più le immagini sono chiare e gli appunti dettagliati, più questo aiuta a velocizzare tutto ciò che c'è da fare nel momento in cui ci si trova sul set, a partire dal posizionamento delle luci, così come quello degli attori, fino alle caratteristiche della camera da utilizzare in quella scena. Inoltre, è anche un ottimo indicatore su tutto ciò che è necessario avere per girare una determinata inquadratura, oltre che indicarne l'ordine e il numero. Trova la sua utilità anche in fasi precedenti rispetto a quando ci si trova sul set: ad esempio, un buon storyboard può essere utilizzato per presentare la propria idea ad un produttore, per confrontarsi con il direttore di fotografia sulla realizzabilità di un'inquadratura, oppure per capire quante inquadrature serviranno per girare una scena e di conseguenza stabilire un piano di lavorazione e un ordine del giorno.

C'è comunque chi non realizza lo storyboard per i propri lavori, considerandolo uno spreco di tempo o di soldi, o chi, come alcuni registi, preferisce trovare l'ispirazione direttamente sul set.

Questo rende lo storyboard utile in base alle necessità, slegandosi quindi anche dal contesto puramente cinematografico. Infatti, viene utilizzato anche in campo pubblicitario, così come nell'intrattenimento, in grandi eventi o spettacoli.

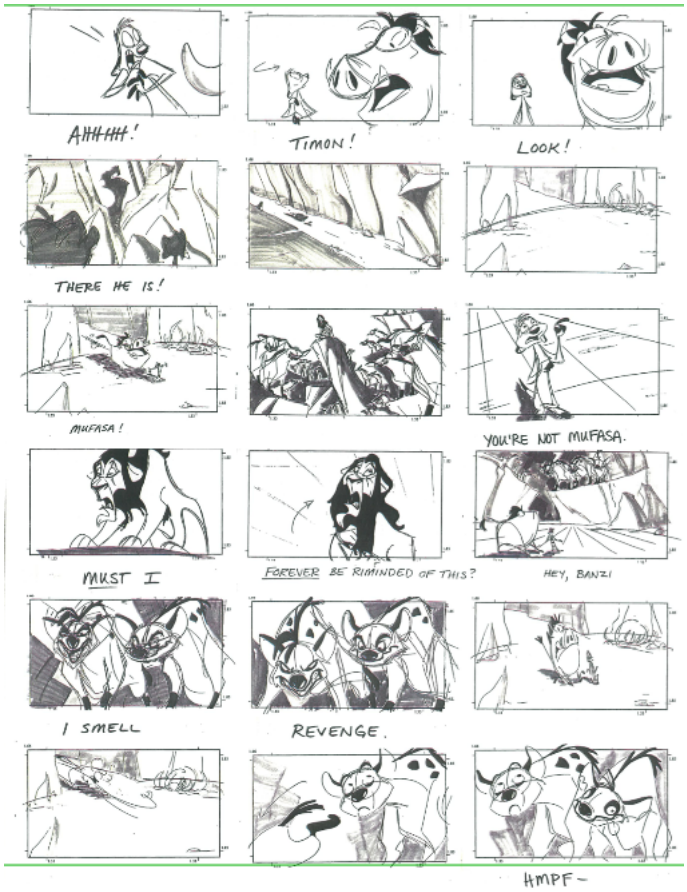


Figura 1.1: Parte dello storyboard de “Il Re Leone” [2]

In questa tesi, però, ci si riferisce alla sua forma cinematografica, della quale saranno analizzate delle linee guida in seguito. Nel suddetto campo, lo storyboard trova la sua origine grazie a Walt Disney che ne fece utilizzo per i suoi corti animati. Uno dei suoi collaboratori, Webb Smith, già nel 1927 per la serie “Oswald il coniglio fortunato”, realizzò lo storyboard completo dei suoi cortometraggi, che ai tempi veniva chiamato “continuity sketches”. Dagli anni ‘30, probabilmente visto il successo ottenuto da Walt Disney, la maggior parte dei film prodotti dagli studios ne faceva uso, così da permettere alla produzione stessa di avere tutti i lavori sotto controllo [1].

1.1 Obiettivo della tesi

La realizzazione di uno storyboard è un processo che, se svolto nella maniera classica, richiede una quantità di tempo significativa, che cresce all’aumentare

della durata del prodotto video che si sta realizzando. Se poi si vogliono avere delle tavole ben fatte e dettagliate, oltre alla necessità di avere un buon storyboard artist, questo tempo aumenta ancor di più.

L'obiettivo di questo progetto di tesi è quindi quello di ridurre queste tempistiche al minimo, di realizzare delle tavole di qualità e, soprattutto, di rendere tutto questo processo il più automatico possibile. In particolare, a partire da una scena animata direttamente dall'utente, il sistema deve interpretarla e generare automaticamente la descrizione testuale, tipica degli storyboard.

1.2 Linee guida storyboard

Per la realizzazione di uno storyboard si utilizza una sequenza di rettangoli (in genere in proporzione 16:9) sotto ai quali è lasciata qualche riga per aggiungere dettagli testuali.

All'interno del rettangolo si crea la rappresentazione dell'inquadratura che si vuole realizzare: pur non essendo necessaria una particolare qualità dei disegni, è importante che essi siano rappresentativi e che facciano capire espressioni e pose dei personaggi, oltre che quale porzione di ambiente è inquadrato tramite la prospettiva.

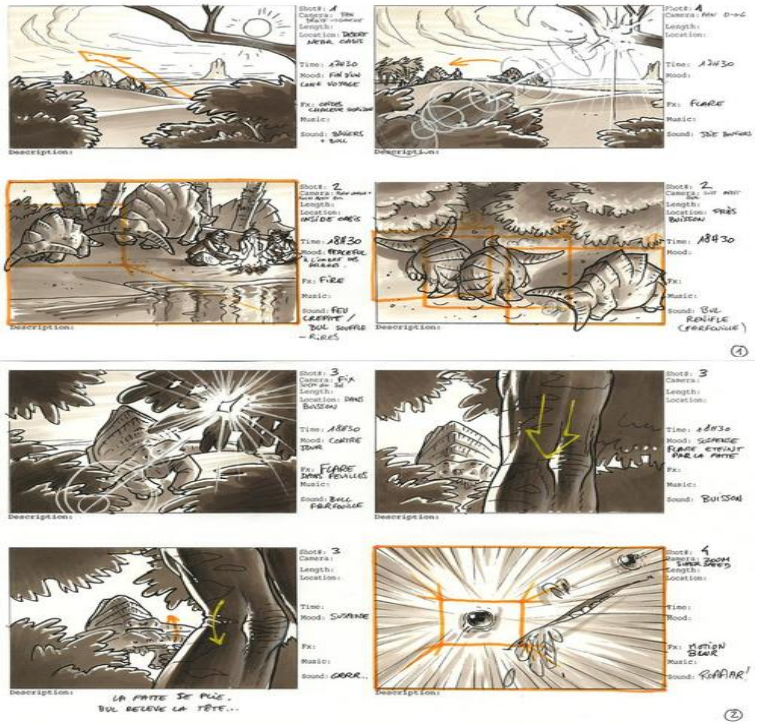


Figura 1.2: Storyboard del videogioco Ryzom [3]

All'interno dei rettangoli si usano delle frecce per indicare i movimenti dei personaggi, mentre altre, anche esterne, vengono poste per indicare i movimenti di camera (carrellata, close-up) (figura 1.2).

Al di sopra del rettangolo si scrive il numero della scena e della relativa inquadratura (non li si trovano spesso, ma possono essere molto utili), mentre al di sotto si appuntano tutte le informazioni necessarie o utili: si possono quindi inserire i movimenti della macchina da presa da realizzare in quello shot, il tipo di camera e relativo obiettivo, luci da usare, descrizione della scena e infine, se presenti, i dialoghi. Se ci sono transizioni tra due inquadrature, le si appuntano in mezzo ad esse.

Nei più dettagliati, tra le annotazioni ci può essere il costo dell'inquadratura, mentre per particolari tipi di riprese come una panoramica o una carrellata si può utilizzare un rettangolo più lungo di modo da farci stare tutti i movimenti al suo interno.

Oltre all'aspetto tecnico, però, ci sono altri elementi fondamentali per la realizzazione di uno storyboard, sia esso per un video o per un prototipo in generale. Nel loro studio [4], Truong, Hayes e Abowd, hanno analizzato diversi storyboard sia realizzati da professionisti sia da amatori, identificando cinque elementi cardine:

- livello di dettaglio, ovvero quanto si decide di disegnare all'interno del singolo frame;
- inclusione del testo, sia se metterlo o meno, sia se metterlo esternamente al frame o all'interno, ad esempio all'interno di nuvolette;
- inclusione di persone ed emozioni, quest'ultime per essere esplicitate a volte necessitano della presenza di persone, altre volte dell'assenza di esse;
- numero di frame, variabile in base al fine cui è destinato uno storyboard, se per un prototipo bisogna stare tra i 3 e i 6 per ogni azione che esso può compiere;
- passaggio del tempo, che può essere esplicito o realizzato tramite transizioni.

A partire da questi elementi e tramite un'analisi dettagliata di come il loro utilizzo, in un verso o in un altro, possano portare alla definizione di uno storyboard funzionale o meno, hanno definito delle linee guida per una creazione che consideri tutto il necessario:

- essere a conoscenza del background culturale di coloro che usufruiranno dello storyboard, di modo da poterlo rendere accessibile in base alle conoscenze possedute;
- essere creativi e aperti nel pensare alla storia, lavorando in brainstorming con tutto il team;

- spezzare in parti (fino a 5) più piccole ogni filone di storia ed essere in grado di descrivere queste sezioni con una frase. Ognuna di queste frasi viene poi tradotta in un disegno che dovrà contenere del testo appropriato, delle persone (se si prevede un'interazione), il passaggio del tempo (se rilevante) ed un livello minimo di dettagli;

- infine, testare ciò che è stato prodotto.

Sperimentando questa loro analisi hanno ottenuto risultati soddisfacenti: chi ha seguito le loro linee guida, ha prodotto uno storyboard in tempi brevi e con la giusta quantità di dettagli.

Anche Kantola e Jokela identificano, in un'analisi [5] del 2007, degli elementi portanti di uno storyboard a partire da degli esempi usati per film e fumetti, e sono:

- contesto, ovvero dove l'azione ha luogo;

- utente/i, cioè coloro i quali vedranno il proprio lavoro affetto dalla presenza del prodotto descritto dallo storyboard;

- obiettivi dell'utente, quindi ciò che risponde alla domanda "perché si sviluppa questa storia?";

- piani, attività mentale dell'utente orientata al trasformare gli obiettivi in comportamenti;

- valutazione, attività mentale orientata all'interpretazione delle caratteristiche;

- azioni dell'utente, ciò che fa l'utente;

- eventi del sistema, azioni o reazioni esterne prodotte dal sistema.

Nel realizzare uno storyboard, analizzano i sopra-citati elementi e sottolineano come il numero di frame che si va a realizzare non sia deciso a priori, ma viene di conseguenza a ciò che man mano si decide di raccontare: in generale, al loro interno, cercano di descrivere l'interazione tra l'utente e il sistema, ma in alcuni il sistema non compare nemmeno, in quanto non è utile per la narrativa.

Il tool COMuICSer [6] sviluppato da Haesen, Meskens, Luyten e Coninx, prende spunto dalle tecniche utilizzate nella realizzazione di fumetti per facilitare lo sviluppo di uno storyboard all'interno di un team eterogeneamente composto. Esso, quindi, aiuta i non tecnici a realizzare qualcosa di utilizzabile dai tecnici.

Le tecniche su cui si basano i fumetti che facilitano lo storyboarding e sulle quali si sono basati nella realizzazione di questo tool sono:

- espressioni facciali, usate per far esprimere emozioni ai personaggi e per suscitarne a chi legge: si basano sulle sei espressioni base, ovvero felicità, dolore, sorpresa, rabbia, serietà e angoscia;
- linguaggio del corpo, usato per esprimere il carattere e il mood del personaggio, ne definisce la personalità;
- personaggi disegnati in sfondi reali, utili per aiutare ad immedesimarsi nel personaggio;
- differenziazione tra i personaggi, basata su tre punti che rendono riconoscibili i personaggi: interiorità, distinzione visiva e tratti espressivi;
- transizioni, basate sul concetto di "chiusura" applicato dal cervello umano, realizzate ponendo vicini due frame di modo da percepirli come uno solo rendendo più immediato il passaggio da un punto ad un altro.

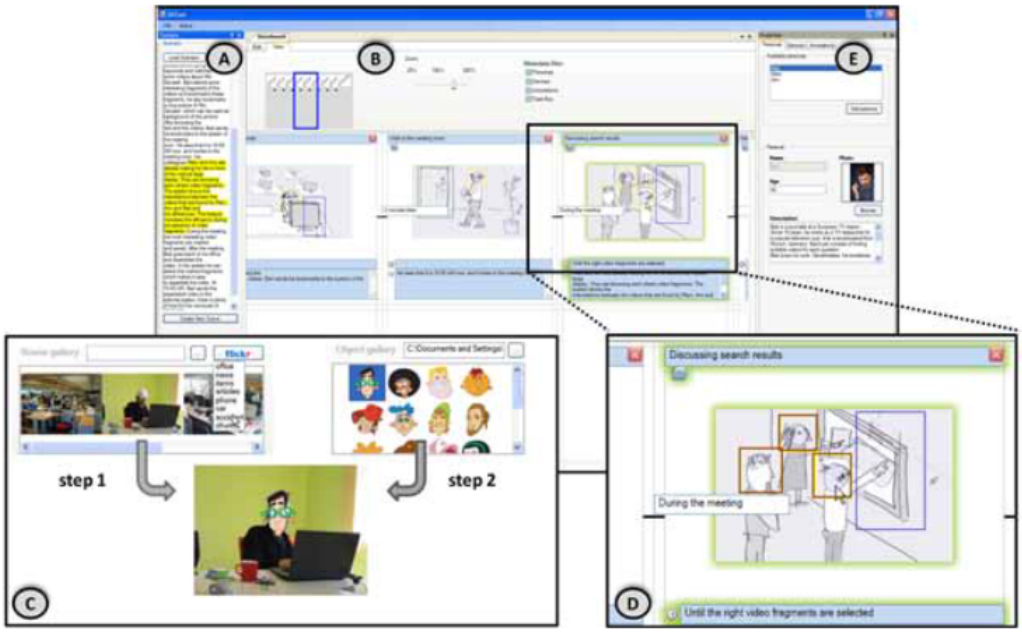


Figura 1.3: Screenshots del tool

All'interno del tool, come si vede in figura 1.3, è possibile inserire uno scenario (A) e creare delle scene nel pannello dello storyboard (b). Queste scene possono contenere degli sketch o una combinazione di fotografie e icone di personaggi (C). Sono supportate anche etichette per le transizioni e informazioni temporali (D) e annotazioni di scena (E).

Un altro lavoro [7] interessante, che porta alla realizzazione di uno storyboard solo con il fine di sperimentare ciò che è stato realizzato, utilizza degli schemi per realizzare la trama di una storia. A partire dalle relazioni tra eventi, danno la possibilità di aggiungere un tassello alla narrazione, basandosi su ciò che è successo al punto precedente.

Programmando le varie azioni utilizzabili e tutti i sotto rami che portano ad una scelta di un'azione conseguente, hanno utilizzato un tool di storyboarding (PlotBoard) che, conoscendo l'azione selezionata, affianca ad una rappresentazione grafica la rispettiva descrizione testuale (figura 1.4).

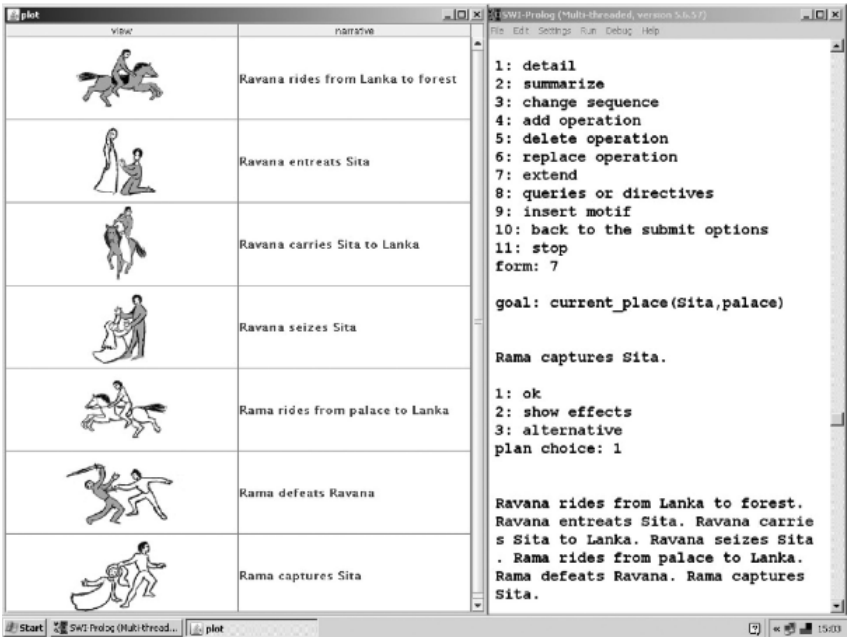


Figura 1.4: Screen di una scena su PlotBoard

Capitolo 2

Stato dell'arte

In questo capitolo si identificano delle modalità più moderne di realizzazione di uno storyboard e, successivamente, viene fatta un'analisi di alcuni articoli e studi fatti il più vicini possibile all'obiettivo di questa tesi. Come si vedrà, però, né nei moderni software né nei risultati ottenuti dagli autori degli articoli, si ritrova il processo che caratterizza questo progetto di tesi: creare uno storyboard a partire da un'animazione realizzata in tempo reale dall'utente. Questa mancanza di materiale da cui prendere spunto, se da un lato ha negato la possibilità di possedere e padroneggiare alcuni concetti utili già in partenza, dall'altro ha permesso una maggiore libertà nella realizzazione della fase di sviluppo, oltre che una più personale realizzazione dell'intero progetto.

2.1 Software per storyboard

Col passare del tempo, si è cercato di rendere più facile e automatico il processo di realizzazione dello storyboard: esistono ad oggi dei software come TVP Animation o ToonBoom Storyboard o siti come storyboardthat.com che ne permettono interamente la creazione, offrendo la struttura base dello storyboard da compilare.

TVP Animation TVP Animation [8] ha un'interfaccia complessa in quanto permette di fare svariate cose oltre agli storyboard (come animatic, animazioni 2D o semplici disegni). Lato storyboard permette il disegno a mano della scena, così come l'aggiunta di campi testuali sotto ogni frame (per farlo bisogna aprire l'apposito pannello). Permette il lavoro per layer così da poter usare uno sfondo uguale su più frame e anche la possibilità di importare immagini proprie. Esiste una versione di prova ed è disponibile per tutte le piattaforme, inoltre fornisce della documentazione nel caso ci si approcci per la prima volta al suo utilizzo.

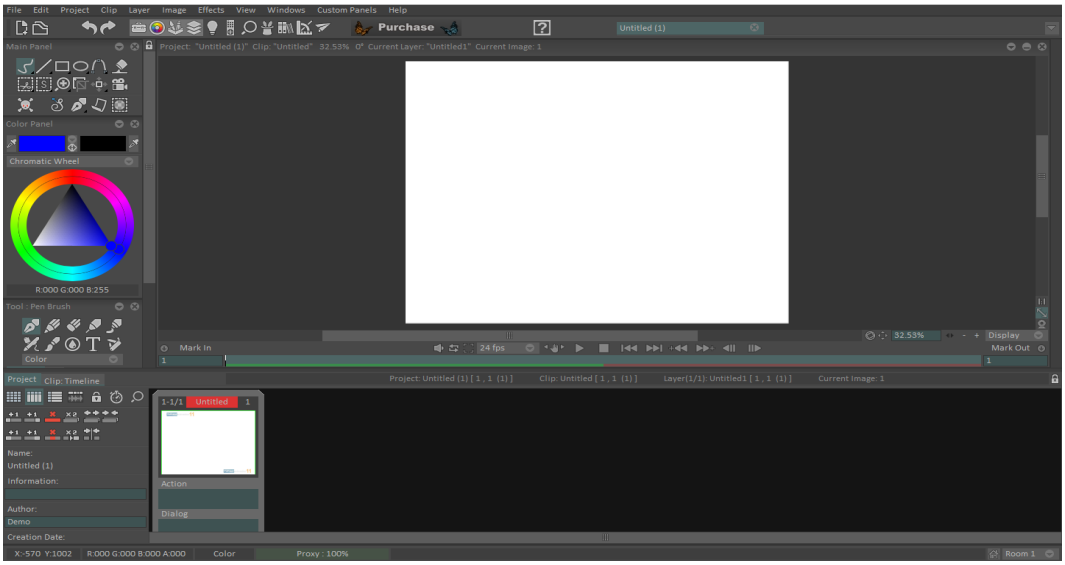


Figura 2.1: Interfaccia TVP Animation

ToonBoom Storyboard ToonBoom Storyboard [9] propone anch'esso molteplici funzionalità e rende la creazione di storyboard ancora più intuitiva del precedente: l'aggiunta dei campi testuali ha un pannello tutto suo nell'interfaccia di base e permette anch'esso l'importazione di immagini sia come layer che come scena. Inoltre, è concesso l'export del progetto anche nella versione di prova, a differenza di TVP Animation.

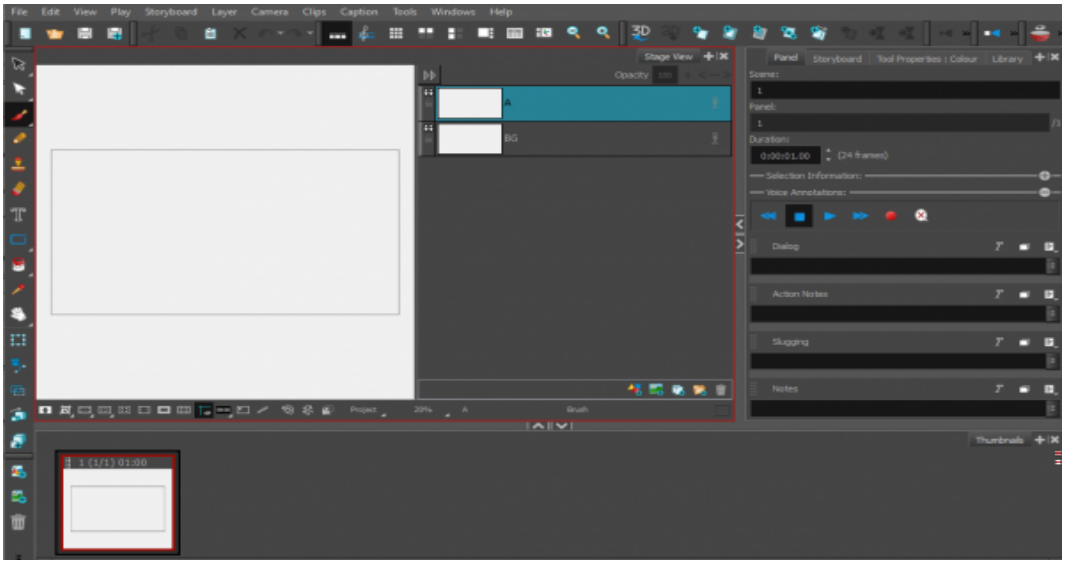


Figura 2.2: Interfaccia ToonBoom Storyboard

Storyboardthat.com Storyboardthat.com [10] invece è uno storyboard creator online che prevede un largo numero elementi già pronti da mettere nella scena (personaggi, ambienti e forme di vario tipo). I personaggi stessi sono a loro volta molto personalizzabili e per questo motivo si pone come strumento molto utile per il suo scopo. È molto intuitivo e facile da utilizzare e permette di ottenere buoni risultati in poco tempo (Figura 2.3). Al contrario dei primi due, non prevede il disegno a mano.

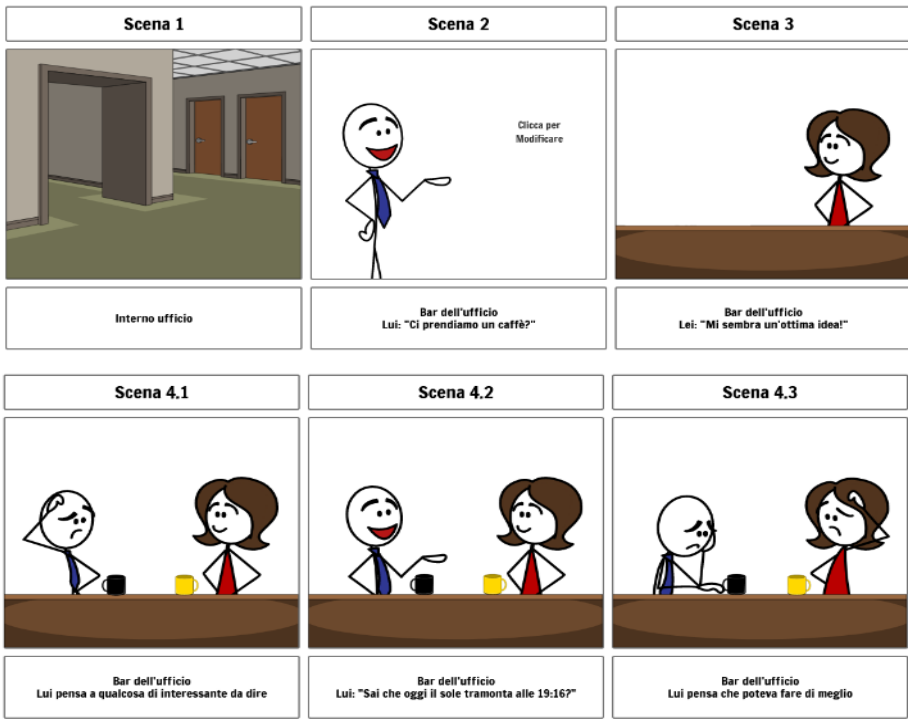


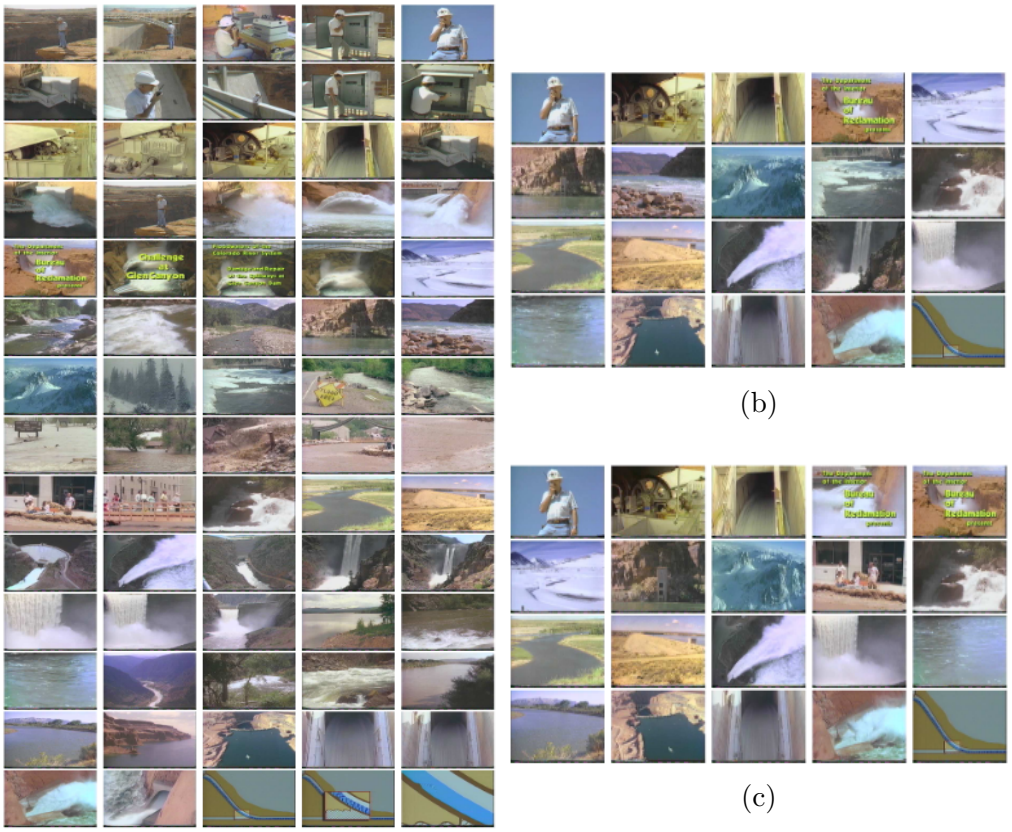
Figura 2.3: Storyboard creato su storyboardthat.com

Esiste poi un altro software, forse il più simile all'applicativo realizzato, che possiede una vasta libreria di oggetti e ambienti, ed è Cine Tracer [11]. Al suo interno è possibile aggiungere e rimuovere personaggi e ambienti a piacimento, i quali possiedono un'alta qualità grazie all'utilizzo di Unreal Engine. Inoltre è possibile gestire tutti i parametri caratteristici della camera, oltre che luci ambientali e simulazioni metereologiche. Tutte queste caratteristiche permettono quindi di catturare delle immagini di estrema qualità e realisticità, che possono poi essere destinate ad uno storyboard. Ciò che non contempla, però, è l'interpretazione delle azioni svolte nella scena, con conseguente creazione della descrizione testuale, a differenza di questo applicativo.

2.2 Ricerche e studi

Ci sono state anche delle ricerche per creare lo storyboard automaticamente, partendo però da un video o un'animazione.

Uno studio [12] del 2009 mette alla base della sua ricerca il fatto che generare uno storyboard selezionando dei frame facenti parte dell'animazione può risultare molto oneroso. Lo schema proposto da Mohanta, Saha e Chanda basato sullo Spanning Tree, crea una sotto-selezione di frame rappresentativi, tenendo in considerazione la diversità che ci deve essere tra i frame selezionati di modo da non avere immagini ripetitive della stessa scena. Hanno anche dimostrato come il loro approccio sia ben funzionante quando vengono selezionati i frame ad una distanza fissa precedentemente decisa (I-frames), i quali poi vengono filtrati per evitare la ridondanza. Si evita quindi di considerare esclusivamente i key-frames che porterebbero ad un aumento del costo computazionale, senza però sacrificare la qualità.



(a) (b) (c)

Figura 2.4: Esempio di utilizzo dello schema basato su Spanning Tree

Nell'esempio da loro pubblicato (figura 2.4), mostrano uno storyboard che rappresenta un intero documentario (a), una versione ridotta utilizzando i key-frames (b), e infine una versione creata tramite I-frames (c).

Nel 2019 invece, alcuni ricercatori di Microsoft Asia in collaborazione con dei componenti della Pechino Film Academy hanno proposto, nel loro "Neural Storyboard Artist: Visualizing Stories with Coherent Image Sequences" [13], una metodologia per creare lo storyboard in maniera automatica a partire da un video, effettuando i loro esperimenti su scene girate in un set e non su video animati. In questo senso, infatti, nel processo di realizzazione dello storyboard, è compresa una fase di eliminazione di parti superflue (per uno storyboard) della scena e di unificazione tra loro dei vari frame realizzati (come, di fatto, sono gli storyboard realizzati a mano). L'unica fase rimasta manuale all'interno di questo processo è la sostituzione della scena e dei personaggi che fanno parte dell'inquadratura e, dopodiché, la selezione della camera che renderizzerà il modello 3D nell'immagine in 2D.

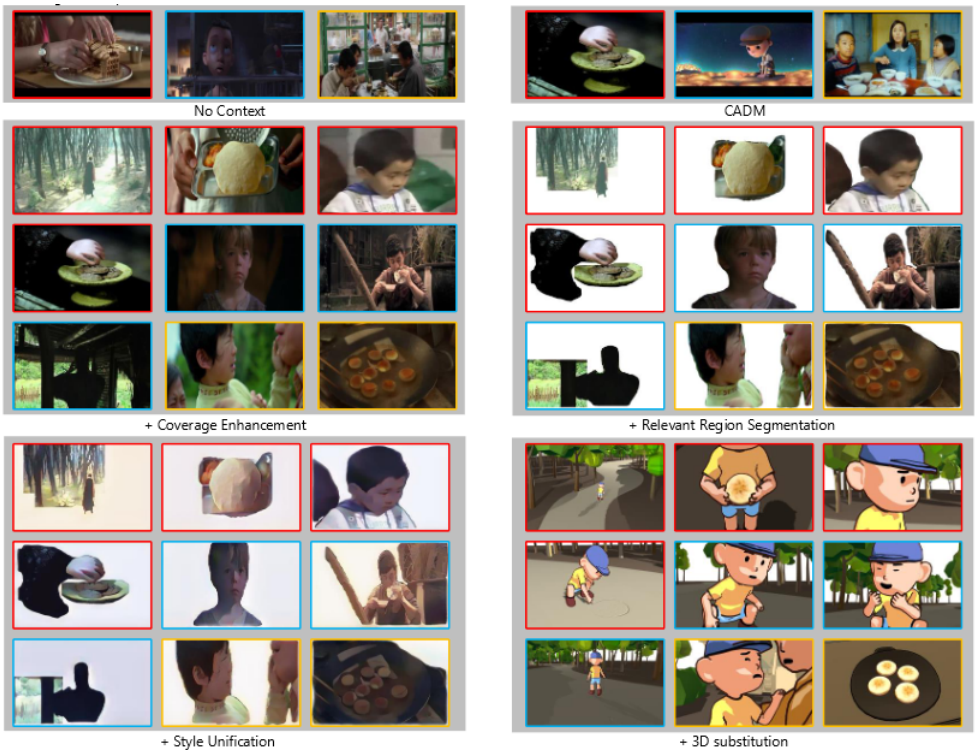


Figura 2.5: Fasi del processo che porta allo storyboard finale

Altri studi invece, si sono focalizzati sul creare animazioni in maniera automatica, partendo da uno script.

Molto importante e innovativo è il lavoro [14] di Held, Gupta, Curless e Agrawala del 2012 ("3D Puppetry: A Kinect-based Interface for 3D Animation"). Esso consiste inizialmente nel riprodurre dei piccoli oggetti reali all'interno dello spazio 3D tramite inquadratura del kinect e un apposito software che ne realizza il modello 3D, e successivamente di muovere fisicamente questi oggetti che, sempre inquadrati dal kinect, vengono mossi in tempo reale anche nell'ambiente 3D, dando quindi la possibilità di realizzare una estremamente personalizzabile animazione. Inoltre, questo sistema permette anche di modificare camera e luci prima, durante e dopo la performance. Marcando i vari oggetti reali, è quindi in grado anche di rilevare collisioni tra le loro rappresentazioni 3D nell'ambiente virtuale. Infine, utilizzando un approccio a livelli dà la possibilità di animare diversi personaggi che si muovono nello stesso momento.

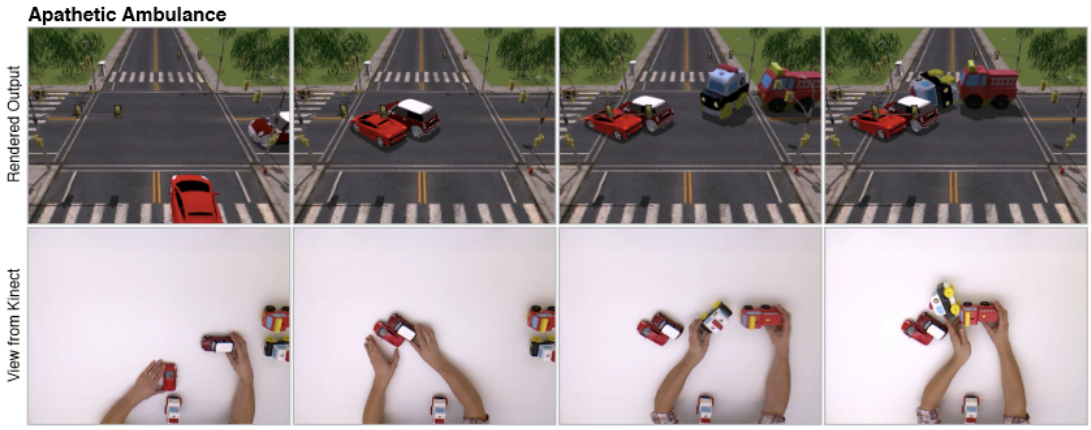


Figura 2.6: Esempio del sistema in azione

Nella figura 2.6, la riga inferiore mostra il performer che manipola gli oggetti fisici, mentre la riga in alto mostra il risultato del sistema che tiene traccia degli oggetti in tempo reale e li riproduce nell'ambiente virtuale.

Più completo, ma che utilizza il processo inverso è lo studio [15] del 2018 ("TakeToons: Script-driven Performance Animation") di Hariharan Subramonyam, Wilmot Li, Eytan Adar e Mira Dontcheva i quali hanno sviluppato un applicativo in grado di animare in automatico a partire da uno script/copione. Tramite determinati comandi vocali, si può iniziare la registrazione di una battuta sul copione e in automatico il personaggio che la sta dicendo viene animato. In sostanza, uno script deciso in precedenza è sensibile alla registrazione di battute che quando vengono "chiamate" provocano una conseguente animazione. Si tratta in sostanza di una sorta di doppiaggio in real-time di un'animazione (negli esempi portati, esclusivamente in 2D).

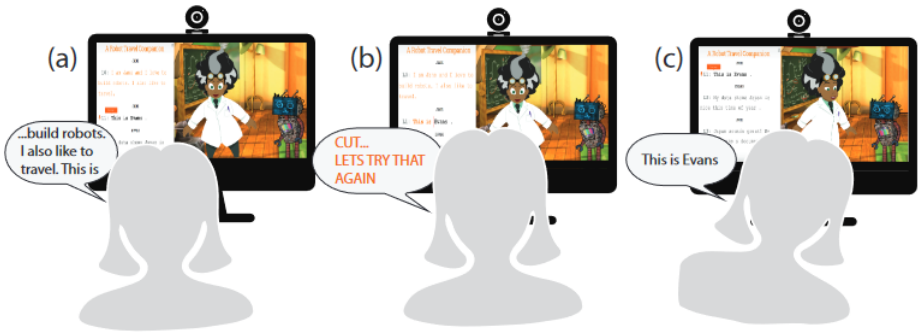


Figura 2.7: Funzionamento dell'applicativo

Come si vede dall'esempio in figura 2.7, la performer legge una riga (a), il testo si evidenzia e scorre automaticamente alla riga successiva. La performer si è dimenticata di muovere la testa a destra, quindi dice: "taglia! proviamola di nuovo" (b) per registrare nuovamente la riga (c).

Sempre su questa linea è il lavoro [16] sviluppato da alcuni ricercatori Disney nel 2019 ("Generating Animations from Screenplays") che, considerando l'esistenza di programmi in grado di gestire solamente frasi semplici, hanno ampliato questo "dizionario" di input testuali per dare più varietà e libertà di espressione per generare un'animazione. Inoltre, questo è uno dei rari studi in cui viene poi considerata la generazione di uno storyboard a partire dalle frasi semplificate prese in input. Anche in questo caso però, si parte da uno script per raggiungere l'animazione.

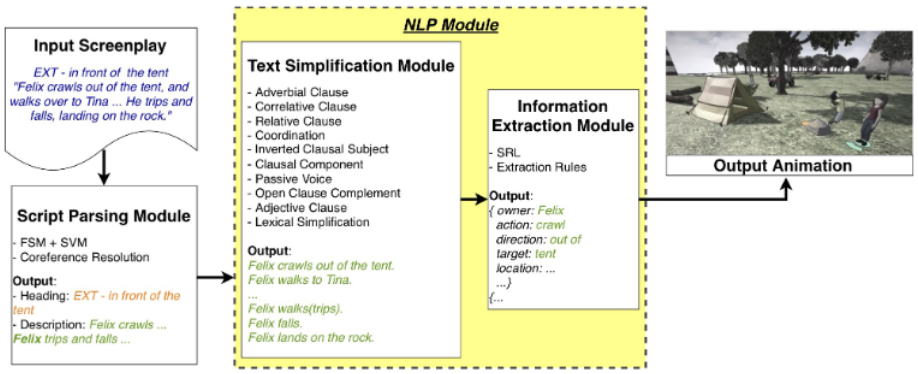


Figura 2.8: Architettura del sistema: segmentazione in diversi blocchi funzionali. Le frasi vengono semplificate e utilizzate per generare l'animazione

Capitolo 3

Tecnologie

In questo capitolo verranno illustrati i software e il linguaggio di programmazione utilizzato, con annesse le librerie necessarie per il funzionamento del sistema.

3.1 Blender

Blender è il software scelto per la realizzazione dell'applicazione in quanto è molto versatile; infatti, tutte le fasi di sviluppo sono state realizzate al suo interno, a partire dalla modellazione fino ad arrivare alla scrittura del codice python che permette il funzionamento.

Questo è un software libero e multipiattaforma di modellazione, rigging, animazione, compositing e rendering di immagini tridimensionali. Dispone inoltre di funzionalità per mappature UV, simulazioni di fluidi, di rivestimenti, di particelle, altre simulazioni non lineari e creazione di applicazioni/giochi 3D.

Blender è dotato di un robusto insieme di funzionalità paragonabili, per caratteristiche e complessità, ad altri noti programmi per la modellazione 3D come Cinema 4D, 3D Studio Max e Maya. Tra le funzionalità di Blender vi è anche l'utilizzo di raytracing e di script (in Python); [17] quest'ultima fondamentale per lo sviluppo di questa tesi.

Inoltre, vengono rilasciate continuamente nuove versioni, nelle quali, rispetto alle precedenti, vengono effettuati cambiamenti dell'interfaccia (le quali rimangono sempre personalizzabili) o vengono aggiunte o tolte delle funzionalità; proprio per quest'ultimo motivo, le versioni utilizzate nella realizzazione delle varie fasi del processo sono state due: la 2.79 e la 2.9.

3.1.1 Versione 2.9

In questo ambiente è stata sviluppata la prima fase del progetto, ovvero la modellazione, la creazione dei materiali, il rigging, l'animazione e la realizzazione degli animatic dei tre casi d'uso.

La modellazione dei personaggi è stata sviluppata per estrusione a partire da un piano, seguendo delle references. Allo stesso modo sono stati realizzati gli altri elementi delle scene, partendo però da oggetti semplici (es. cubo) per poi scalarli e adattarli.

La creazione e conseguente assegnazione dei materiali è stata fatta, per gli oggetti con più materiali, per gruppi di vertici.

Il rigging, richiede una fase preliminare nella quale si rinominano tutti i *bones* creati in maniera corrispondente alla porzione di corpo che andranno a controllare. Effettuato questo passaggio, è stata creata la parentela tra armatura e mesh con la modalità *with automatic weights*, che associa in maniera automatica un determinato gruppo di vertici ad un osso dell'armatura, in base alla posizione dell'una rispetto all'altra. Questo step aiuta in quanto permette di non partire da zero nell'assegnazione dei vertici, ma non sempre risulta preciso: si è poi passati ad una fase manuale in cui sono stati disassociati eventuali vertici di troppo e associati quelli mancati dall'operazione automatica del software.

Fatta eccezione per l'animazione della camminata, tutte le altre action sono molto semplici e non hanno avuto bisogno di particolari references per essere realizzate (applauso per la ragazza, abbaiare o dare la zampa per il cane, belare per la pecora). I diversi cicli di camminata sono stati realizzati seguendo le indicazioni [18] di Richard Williams (direttore dell'animazione di "Chi ha incastrato Roger Rabbit?"), il quale spiega frame by frame tutte le pose che, umanoidi o animali, assumono per completare il ciclo.

Questo è reso possibile in Blender grazie agli strumenti di animazione in esso contenuti. Selezionando l'armatura ed entrando nella *pose mode*, è possibile creare un keyframe della posa in quel momento; spostandosi avanti nella timeline, cambiando la posa e creando un nuovo keyframe, Blender interpola i cosiddetti *in-between frames*, ovvero tutti i frame in mezzo ai due keyframe precedentemente salvati. In questo modo si crea un'animazione dal primo al secondo frame, e avanti così fino al termine dell'animazione desiderata.

Terminata un'animazione, Blender permette di creare delle action, ovvero un unico elemento che contiene i keyframes prima creati e che può essere ricercato tramite script. In associazione ad essa viene creata una *track* la quale può contenere più action. Nell'ambito di questa tesi però, ogni track conterrà una sola action. Sono state quindi definite delle librerie di action per ogni personaggio.

L'animatic è una sorta di storyboard in movimento, utililzzato nel campo dell'animazione: non sono necessari i personaggi definitivi o le action, in quanto il suo

scopo è quello di dare un'idea preventiva di ciò che si sta realizzando per capire se può funzionare, focalizzandosi sui tempi delle varie inquadrature e sui movimenti di camera.

La scelta di sviluppare le suddette fasi in questa versione è stata puramente per familiarità con la stessa, infatti tutte queste operazioni sono realizzabili in tutte le altre versioni. Inoltre, altra qualità del software, tutto ciò che qui è stato prodotto è stato trasferito sulla 2.79 senza alcuna perdita di qualità o dettagli.

Figura 3.1: Personaggi modellati nella versione 2.9

3.1.2 Versione 2.79

Perché, dunque, utilizzare anche un'altra versione? La 2.79 è l'ultima ad avere al suo interno il game engine, ovvero un'intera suite dedicata alla realizzazione di contenuti interattivi in tempo reale. Questo motore di gioco (BGE) è stato scritto da zero in C++ come componente sostanzialmente indipendente, e include il supporto per funzionalità quali lo scripting Python e il suono OpenAL 3D. [19]

Per sfruttare al meglio le possibilità e potenzialità del BGE, bisogna utilizzare tre componenti, interconnettendole tra loro: queste sono i logic bricks, le game properties, visualizzabili e manipolabili nel Logic Editor, e gli script Python, da scrivere nel Text Editor (figura 3.2). L'unione e il corretto utilizzo di queste parti, permettono di realizzare dei mini giochi con vere e proprie meccaniche di gioco. Nello sviluppo di questa tesi, sono state utilizzate con lo scopo di comandare i

personaggi e realizzare immagini tramite le camere, al fine di realizzare la scena e il conseguente storyboard pensati in precedenza.

I logic bricks sono di tre tipi: *sensori*, *controllori* e *attuatori*.

I sensori restituiscono un output quando accade qualcosa, come ad esempio una collisione tra due oggetti, un tasto premuto sulla tastiera o la vicinanza ad un oggetto con determinate caratteristiche. Quando un sensore viene attivato, manda un segnale positivo al controllore a cui è connesso nel Logic Editor. [20]

I controllori sono coloro che utilizzano i dati ricevuti dai sensori per effettuare le operazioni specificate e dare un segnale agli attuatori collegati. [21] Essi possono essere dei semplici operatori logici (AND, OR ecc.) o dei veri e propri script tramite il quale utilizzare in modo più elaborato ciò che viene mandato dai sensori.

Gli attuatori realizzano delle azioni, come il movimento o la creazione di oggetti ed effettuano il loro compito nel momento in cui ricevono un segnale positivo dal/dai controllori a cui sono legati. [22]

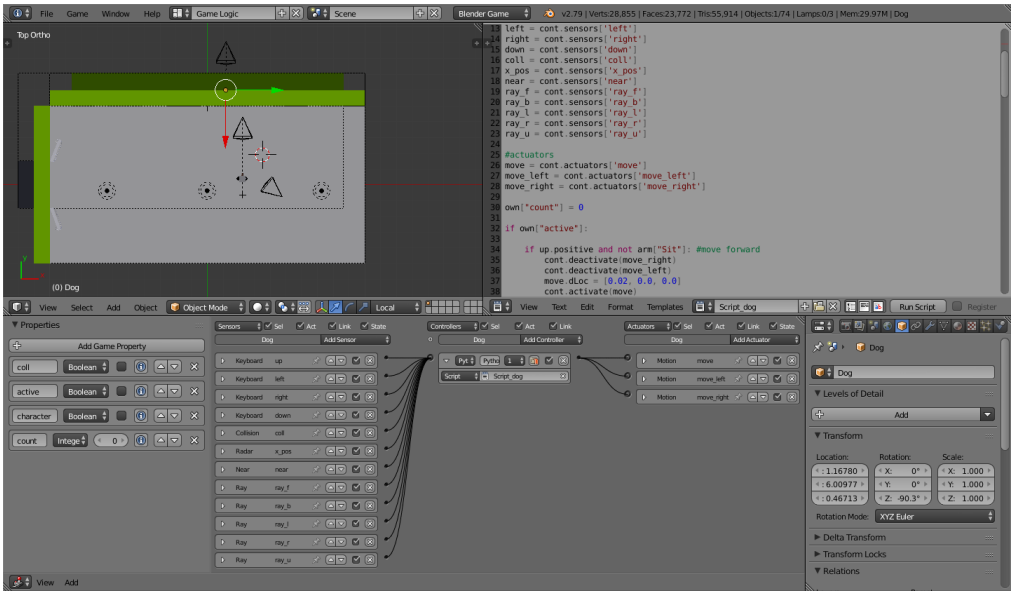


Figura 3.2: Ambiente di lavoro con la v2.79 con logic bricks e script Python

Ognuno dei tre tipi di logic bricks possiede a sua volta un buon numero di tipi di, rispettivamente, sensori [23], controllori [24] e attuatori [25]. Non essendo stati utilizzati tutti nello sviluppo della tesi, verranno qui di seguito introdotti solo quelli resi necessari per il funzionamento del sistema di modo da rendere più comprensibile il sottocapitolo 4.3 riguardante il come questi siano stati utilizzati.

Sensor options Tutti i sensori hanno alcune funzionalità in comune. Dalla figura 3.3 si vede come ci siano cinque bottoni che sono quelli posseduti da tutti i sensori. I primi due innescano i controllori connessi finché il sensore ha

valore, rispettivamente, True e False. *Level* è sensibile al cambiamento di stato, mentre *Tap* pone lo stato del sensore a negativo un frame dopo esser diventato positivo. Infine *Invert* cambia l'output con il suo opposto.

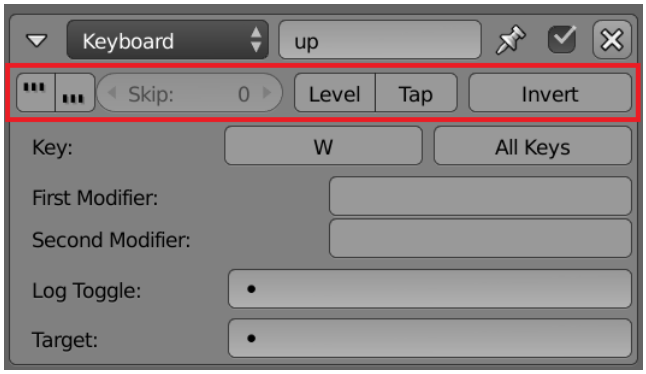


Figura 3.3: Evidenziata la parte comune a tutti i sensori

Always Sensor Sensore che dà un continuo segnale in output, usato quando è necessario effettuare un'azione ripetutamente o ogni determinati intervalli [26].

Collison Sensor Sensore che si attiva al contatto con un oggetto. Se si specifica una game property o un materiale, si attiva solo se l'oggetto la/lo possiede [27].

Delay Sensor Serve per ritardare un evento o per determinare la durata dello stesso. Questo grazie a i due parametri modificabili *Delay* e *Duration* [28].

Keyboard Sensor Usato per percepire gli input da tastiera, se ne può specificare uno nell'apposita casella [29].

Mouse Sensor In grado di rilevare eventi legati al mouse, se ne deve specificare uno tra movimento e tasti presenti [30].

Near Sensor Intercetta oggetti i quali devono avere *Actor* attivo nel pannello della fisica. Può filtrare questi oggetti tramite una game property, se specificata. Ha un raggio d'azione modificabile il quale crea una sfera intorno al centro dell'oggetto [31].

Property Sensor Rileva cambiamenti di valore di una game property posseduta dall'oggetto che utilizza questo sensore [32].

Radar Sensor Molto simile al near sensor, al posto di una sfera si crea un cono con la punta nel centro dell'oggetto, con direzione specificata e con la base ad una distanza anch'essa specificata. Anche l'angolo del cono creato è personalizzabile. Anche in questo caso, rileva gli oggetti con *Actor* attivo e, specificandola, che possiedono una determinata game property [33].

Ray Sensor Crea un raggio nella direzione e di una lunghezza specificate. Manda un segnale positivo quando incontra un oggetto con una game property o un materiale, se specificata/o. Inoltre, ha un tasto denominato *X-Ray Mode* che, se attivato, permette al raggio di passare attraverso gli oggetti che non possiedono l'elemento specificato [34].

AND Controller Funziona come un operatore logico AND, quindi ha un output True se tutti i suoi input hanno valore True, False se uno solo degli input è False [35].

OR Controller Funziona come un operatore logico OR, quindi ha un output True se almeno un input ha valore True, False se tutti gli input sono False [36].

Python Controller Questo controllore esegue uno script ogni qual volta viene attivato dal/dai sensore/i connesso/i. Può compilare uno script per intero (tramite il nome) scegliendo la modalità *Script* o un solo modulo al suo interno (tramite nomescript.nomemodulo) con la modalità *Module* [37].

Edit Object Actuator Permette di modificare le impostazioni di un oggetto in-game. Può sostituire una mesh, aggiungere oggetti o distruggerli [38].

Mouse Actuator Lo si può utilizzare per mostrare/nascondere il cursore o per controllare la rotazione dell'oggetto, specificando parametri come la sensibilità e l'angolazione massima e minima [39].

Motion Actuator Mette in movimento l'oggetto, sia spostandolo sia ruotandolo, secondo valori specificati [40].

Property Actuator Se attivato, questo attuatore è in grado di cambiare il valore della game property specificata [41].

Visibility Actuator Permette di cambiare la visibilità dell'oggetto in-game [42].

Per quanto riguarda le game properties, sono state utilizzate quelle di tipo *integer*, che utilizza interi tra -10000 e 10000, *boolean*, con valori True e False, *string*, che può contenere fino a 128 caratteri e *timer*, che tiene conto del tempo che scorre finché l'oggetto esiste.

3.2 Python

Come accennato in precedenza, nonostante il BGE sia programmato in C++, il linguaggio richiesto dagli script al suo interno è Python.

Python è un linguaggio di programmazione ad alto livello e orientato ad oggetti. Inoltre è pseudocompilato: un interprete analizza il codice sorgente e, se corretto, lo esegue. Non esiste infatti una fase di compilazione separata che generi un file eseguibile (come in C). Tutto ciò lo rende un linguaggio portabile: è quindi eseguibile sia su Mac, Linux o Microsoft, a patto di avere una versione adatta dell'interprete [43].

3.2.1 Librerie utilizzate

Diverse librerie di Python sono state importate e utilizzate nel codice (tramite il comando *import nomelibreria*), ed esse sono: *bpy*, *bge*, *math*, *re*, *time*, *base64*, *webbrowser* e *os*.

bpy Modulo che dà accesso ai dati, alle classi e alle funzioni di Blender [44]. Necessario se si intende lavorare in Blender in modo "schematico" tramite script e non in modo più "creativo" tramite mouse e tastiera. Ad esempio, permette di creare geometrie, posizionandole dove si preferisce e modificandone le proprietà e nel momento in cui il codice viene eseguito, la viewport 3D viene aggiornata. Non è stata molto utilizzata in questa tesi in quanto non si è reso necessario l'accesso alle caratteristiche degli oggetti, fatta eccezione per il recupero e l'utilizzo delle *nla_tracks*, ovvero delle tracce consistenti in una (ad esempio, l'animazione di una camminata) o più (camminata seguita da capriola) *action* messe insieme.

bge Modulo cardine nell'utilizzo appunto del BGE, è il bge.logic che permette l'accesso a determinate funzioni [45]. Grazie a questa libreria si riesce a creare un ponte tra i vari *GameObjects* della scena e gli script utilizzando due funzioni fondamentali: *getCurrentScene()*, dalla quale si possono estrapolare tutti gli oggetti in scena, e *getCurrentController()*, che porta al controllore a cui è assegnato lo script nel quale viene chiamata questa funzione e dal quale si possono ottenere i sensori, gli attuatori e l'oggetto al quale sono assegnati i relativi logic bricks. Da quest'ultimo, si può poi accedere a tutte le game properties. Sempre dal modulo logic, viene utilizzata anche *endGame()* che ha il compito di terminare il gioco. Inoltre, in questa tesi, viene utilizzato anche il bge.render, che tramite la funzione *makeScreenshot()* fa una cattura dell'inquadratura in quel momento.

math Questa libreria dà accesso a tutte le funzioni matematiche definite nella libreria standard del C [46]. Nella tesi vengono utilizzate due funzioni: *sqrt()*,

che calcola la radice quadrata dell'argomento passato nella parentesi, usata nell'ambito del calcolo dell'orientamento di un oggetto rispetto ad un altro, e *ceil()*, che arrotonda all'intero successivo, usata durante la ricerca di sinonimi effettuata per togliere le ripetizioni nella frase arrotondando il rapporto tra il numero di ripetizioni e il numero di alternative di quell'espressione presenti nel vocabolario definito.

re Con questo modulo sono fornite funzioni di confronto e ricerca sulle *RegEx*, ovvero sequenze di caratteri [47]. In questo lavoro è stata utilizzata la funzione *compile()* per cercare, nella stringa che descrive l'inquadratura realizzata, i segni di punteggiatura che delimitano una frase, scomponendo la suddetta stringa nelle varie frasi che la compongono e trasformando in maiuscolo il primo carattere di ogni frase.

time Libreria che fornisce funzioni legate al tempo [48]. Nel progetto è stata utilizzata la funzione *sleep()*, la quale sospende il programma per il tempo specificato, nel momento in cui si salva l'immagine desiderata da destinare allo storyboard: senza questa pausa forzata, il programma non riesce a trovare l'inquadratura appena salvata.

base64 In questa libreria sono presenti funzioni per codificare dati binari in codice ASCII e per decodificare quest'ultimo e tornare al binario [49]. Nella tesi viene usata la funzione *b64encode()* per codificare l'immagine salvata e poi decodificarla di modo che sia leggibile dal file .html su cui verrà realizzato lo storyboard.

webbrowser Con questo modulo è possibile mostrare documenti web all'utente [50]. In questo progetto è utilizzata solamente la funzione *open_new()*, la quale si occupa di aprire, in una nuova scheda del browser, il file passato come parametro.

os Tramite questa libreria è possibile utilizzare funzionalità dipendenti dal sistema operativo [51]. Il sistema utilizza la funzione *chdir()* che permette di cambiare la cartella in cui vengono salvati i file prodotti.

3.2.2 Creare una propria libreria

Nonostante le librerie di Python, e le relative funzioni, siano molteplici, è possibile e, per la realizzazione di questo progetto, necessario, creare una propria libreria con al suo interno tutte le funzioni necessarie. Questo passaggio permette di non dover scrivere le stesse linee di codice per ogni personaggio nella scena e, oltre a snellire i vari script realizzati, riduce anche i tempi di realizzazione di una nuova storia

con nuovi interpreti e diverse ambientazioni, in quanto, se scritte bene, a queste funzioni basterà passare come parametri i nuovi personaggi e elementi presenti.

Per realizzarla, basta aprire il *Text Editor* e salvare un nuovo file con estensione .py. Al suo interno si definiscono le funzioni con la struttura seguente:

```
def function(parameter1, parameter2):

    #function body

    return
```

dove "function" sarà il nome con il quale utilizzarla negli altri script, mentre "parameter1" e "parameter2" i parametri necessari al suo funzionamento. Se la funzione deve solo svolgere dei compiti senza creare un risultato la si chiude con il "return", se invece deve portare un valore si scriverà "return value" dove "value" sarà stato calcolato nel body.

Così come per le librerie predefinite in Python, anche questa dovrà essere importata negli script che intendono utilizzare le funzioni presenti al suo interno. Per rendere più veloce la chiamata delle funzioni, al posto di scrivere import *nomelibreria* e poi successivamente *nomelibreria.function(a, b)*, si può scrivere *from nomelibreria import **, così che poi si scriverà direttamente *function(a, b)* (con "*" si intende "tutte le funzioni presenti al suo interno", in caso si voglia importare una sola funzione basta sostituire "*" con il nome della stessa).

Questa libreria però, non può coprire tutto il codice necessario al funzionamento dell'intero processo, in quanto diverse parti possono variare da storia a storia (dal generale, come cosa possa fare un personaggio, al dettaglio, come in che modo viene nominata una determinata *action*). Ragion per cui è stato realizzato anche un file di testo con le indicazioni su tutto ciò che c'è da fare per far si che le funzioni della libreria svolgano il loro compito correttamente, realizzando al meglio gli altri script necessari.

Capitolo 4

Funzionamento dell'applicativo

4.1 Requisiti

Il sistema è pensato per mettere l'utente nella posizione di raccontare e creare la storia a suo piacimento, in tempo reale.

Questo significa che gli vengono affidati il comando e la gestione di tutti gli elementi controllabili all'interno della scena (personaggi e macchina da presa).

Tramite mouse e tastiera, dunque, l'utente può far effettuare ai personaggi le azioni predefinite, può cambiare elemento controllato, può muovere la macchina da presa all'interno dell'ambiente per cercare la migliore inquadratura (cambiando anche la lunghezza focale, se necessario) e può, nel momento in cui tutto è al posto desiderato, salvare l'inquadratura attuale che verrà destinata allo storyboard. Solo controllando la macchina da presa principale sarà però possibile effettuare il salvataggio dell'immagine, mentre controllando i personaggi è possibile definire azioni contemporanee (effettuando il cambio personaggio mentre un'azione è in corso) o sequenziali (aspettando il termine dell'azione per cambiare personaggio). Durante queste fasi, al sistema è affidato il ruolo di interpretare tutto ciò che avviene nella scena virtuale, comprese le informazioni di contesto, e di tradurle in modo automatico in linguaggio testuale. Le frasi generate, verranno poi allegate all'inquadratura che l'utente ha deciso di catturare, per creare un elemento della sequenza che comporrà lo storyboard finale.

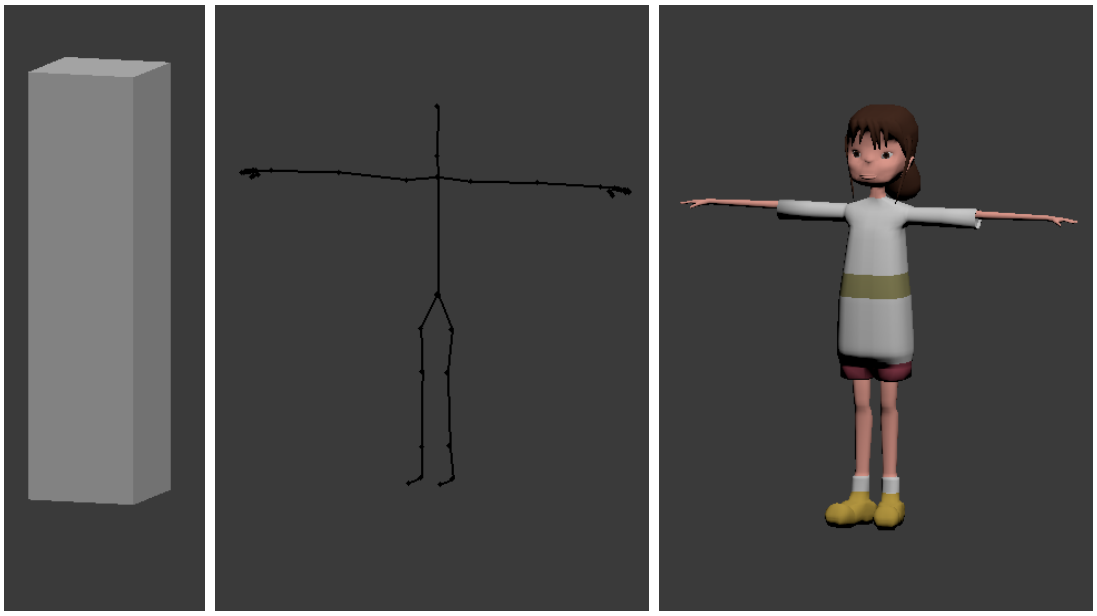
Il sistema, inoltre, contempla anche eventuali errori o volontà di modificare ciò che è stato realizzato, quindi all'utente è anche permesso tornare indietro per rifare un'inquadratura.

4.2 Design

Tutti gli elementi presenti in scena, per far si che il sistema li riconosca e li interpreti in modo corretto, devono essere realizzati secondo un certo criterio. Questo sia nella precedentemente definita prima fase (modellazione, rigging e animazione), sia nella fase successiva quando si approccia il BGE e si rende necessario l'utilizzo, in particolare, delle *game properties* e dei *sensori*.

4.2.1 Personaggi

La struttura di composizione di un personaggio è la più articolata e non si riduce solo alla mesh che lo definisce.



(a) (b) (c)

Figura 4.1: Le 3 componenti essenziali di un personaggio

Come si vede dalla figura 4.1, un personaggio ha, in ordine di realizzazione, una mesh (c), da nominare, ad esempio in questo caso, *Girl_mesh*, la quale sarà riggata e figlia di un'armatura (b), *Girl_arm*, che sarà figlia di un solido semplice, come un parallelepipedo (a), *Girl*. L'ordine di parentela è quindi: a padre di b, che è padre di c.

La mesh è l'elemento estetico del personaggio e ciò che l'utente vedrà muoversi all'interno della scena, ma ad essa non sono associati né script né semplici logic bricks al contrario delle altre due componenti. Infatti, all'armatura saranno associati sensori e script che utilizzano e mettono in pratica le action realizzate, mentre

il parallelepipedo avrà il compito di muovere nello spazio il personaggio e rilevare le informazioni di contesto intorno a sé. Si utilizza un oggetto semplice in quanto un altro compito è quello di rilevare le collisioni, ed avendo pochi vertici rende facile e leggero il calcolo al software. Non si può evitare di usare questo oggetto box, in quanto se per ipotesi si applicasse la fisica *character* anche solo a due mesh complesse (come figura 4.1c), Blender non risponderebbe in modo corretto. Essendo questo applicativo pensato per contemplare un numero n di personaggi, si rende necessaria questa struttura a tre elementi.

C'è anche un quarto elemento importante (che non necessita un nome specifico) utilizzato per rilevare la direzione del personaggio. Qui è stato utilizzato un oggetto empty, reso figlio dell'armatura, e posto alle stesse coordinate locali degli assi y e z, mentre rispetto all'asse locale x è scostato in avanti. La posizione è fondamentale, in quanto sarà utilizzato per calcolare un vettore di direzione che parte dall'armatura e arriva all'empty. Questo verrà poi utiizzato dalla funzione *rayCastTo()* per identificare la direzione del personaggio.

Fondamentale in questa fase, e si vedrà più avanti perché, è la denominazione delle action e rispettive tracks associate all'armatura: entrambe devono essere nominate con la struttura verbo + nome dell'armatura, facendo attenzione anche alle eventuali lettere maiuscole (ad es. per la camminata della ragazza, il nome sarà *WalkGirl_arm*).

Un personaggio ha anche altre componenti che non sono necessarie al funzionamento del sistema (come invece sono le precedenti), ma servono a favorire la fruizione dell'utente (allo stato attuale del codice sviluppato sono contemplate, quindi se non si intende modificare la libreria realizzata, devono essere inserite).

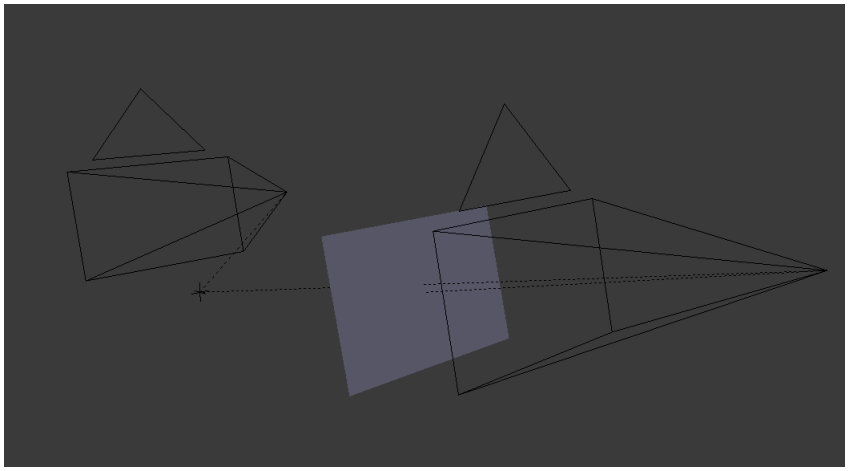


Figura 4.2: Le componenti aggiuntive di un personaggio (camera)

Anche in questo caso è fondamentale denominare questi oggetti nel modo giusto.

Partendo da sinistra nella figura 4.2, si crea un oggetto *empty* che si imparenta all'armatura e gli si assegna il nome di, continuando dagli esempi precedenti, *EmptyGirl_arm*. Essendo figlio dell'armatura seguirà i suoi movimenti e le sue rotazioni e va posizionato circa sul collo. A questo empty si imparentano le due camere presenti, nominando quella di destra *CameraGirl*, mentre l'altra non necessita un nome specifico. Esse seguiranno quindi i movimenti dell'empty padre e la posizione di quella di destra va definita testando l'inquadratura, di modo da avere una buona visione del personaggio, mentre l'altra va posizionata di modo da inquadrare una porzione di testa. Infine il piano presente al centro serve soltanto a mostrare i comandi relativi al personaggio, tramite una texture appositamente creata e applicata; è gestito dalla libreria ma non tramite il nome, non necessita quindi di un nome specifico ma per ordine e riconoscibilità gli si dà il nome *Girl_instr*.

Sono inoltre presenti due frecce, che vengono visualizzate alternativamente, che indicano se il personaggio si sta muovendo verso un punto di interesse o un altro personaggio (verde), o meno (rossa). Se un personaggio lo necessita, è anche presente un cerchio verde posto alla base di un altro personaggio, che viene visualizzato quando ci sono le condizioni per poter compiere una *action* in relazione con esso.

4.2.2 Gruppo di personaggi

Come si vedrà più avanti, nel terzo caso d'uso è stato inserito un gruppo di personaggi (un gregge) da controllare come fossero un unico personaggio, per fare in modo che il sistema fosse in grado di gestire anche questa casistica. La struttura di ogni singolo personaggio del gruppo è la stessa vista al paragrafo 4.2.1 (prime tre componenti).

Ognuno dei box esterni (*Sheep.001*, *Sheep.002*, ecc.) è figlio di un empty denominato *Flock_arm*, inoltre è presente un altro empty *Flock* al quale saranno assegnati alcuni sensori. Le tre componenti aggiuntive sono imparentate tra loro come in precedenza, ma l'empty non è imparentato a nulla non avendo un'armatura del quale copiare movimenti e rotazioni. Per questo, queste sue caratteristiche sono calcolate in tempo reale, prendendo come posizione il centro di massa calcolato su tutti i componenti del gruppo (stessa posizione che viene assegnata all'elemento *Flock*).

4.2.3 Macchina da presa

Oltre alle camere legate ai personaggi, è necessario averne una principale. Questa sarà quella tramite la quale l'utente deciderà angolazione, distanza e lunghezza focale dell'inquadratura da aggiungere allo storyboard. Anch'essa ha dei sensori

per gestirne il movimento all'interno della scena in quanto elemento controllabile al pari dei personaggi, e ha come nome *Camera*. La sua struttura è molto semplice, in quanto costituita dall'oggetto camera in sé e da due oggetti figli, ovvero un piano, come quello visto in precedenza, per mostrare le istruzioni relative ad essa e un oggetto testo che verrà usato per mostrare la didascalia creata e le informazioni della camera (campo e lunghezza focale), quando queste cambiano, per un tempo proporzionale alla lunghezza della stessa.

4.2.4 Altri oggetti

Nella scena sono poi presenti elementi di contesto, che non vengono controllati dall'utente. Questi variano da caso a caso, quindi non sono presenti particolari direttive sulla loro realizzazione. L'unica caratteristica necessaria per far sì che il sistema li rilevi correttamente, è porre tutti gli elementi costituenti un oggetto come figli di un empty che avrà come nome ciò che si vorrà veder scritto nella descrizione della scena.

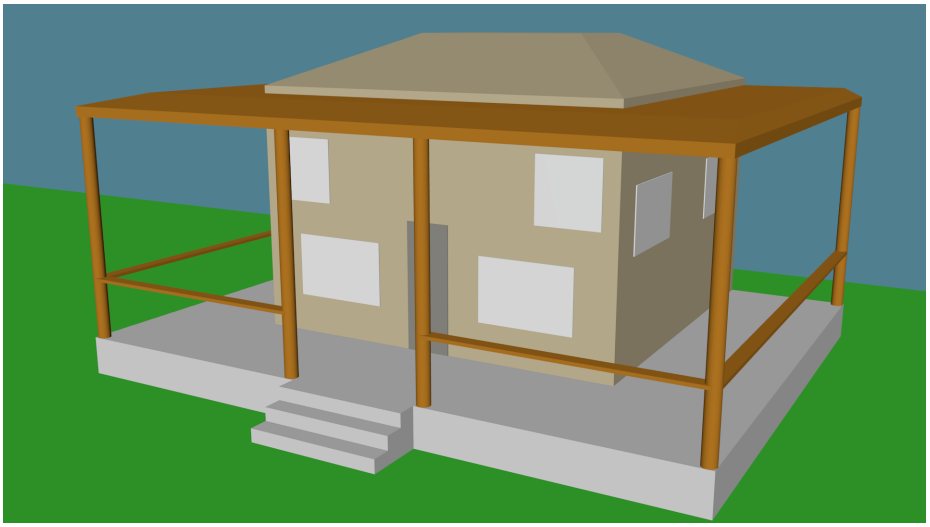


Figura 4.3: Più elementi compongono la casa

Come si vede nella figura 4.3, la casa è formata da diverse parti accostate tra loro e ognuna di esse è figlia di un oggetto empty, in questo caso chiamato *house*.

Quando necessario, in alcune componenti è stato inserito, come per i personaggi, un cerchio verde alla base, che si attiva quando è possibile effettuare ciò che gli è stato designato, o per indicare che il personaggio si trova in punto specifico.

4.3 Sviluppo

Nel momento in cui tutti gli elementi della scena sono completi di tutte le loro componenti, si passa alla definizione, per tutti coloro che lo necessitano, degli script, logic bricks e game properties. Qui verrà indicato tutto ciò che è necessario per il corretto funzionamento dell'applicativo. Ovviamente da caso a caso potrebbero essere utili dei logic bricks in più o uno script aggiuntivo, e per questo motivo verranno indicate anche delle eccezioni che si sono create nello sviluppo degli use cases.

4.3.1 storyboardlibrary.py

Per comprendere al meglio la struttura dei vari script spiegata successivamente, è utile fare prima una breve spiegazione delle funzioni presenti nella libreria creata.

Prima di definirle bisogna importare le librerie necessarie, ovvero *bpy*, *bge*, *math* e *re*, e creare due *storage* nei quali verranno salvate alcune informazioni, definendoli come *objectStorage = {}* e *captionStorage = {}* e un array, *captions = []*, nel quale verrà aggiunta ad ogni inquadratura la descrizione creata.

controlled(base,elements) base: oggetto nella scena che possiede la game property *counter* la quale tiene il conto di ogni scena aggiunta allo storyboard (tornerà utile per molte funzioni), elements: array di liste di oggetti controllabili (personaggi + camera). Ogni lista relativa a un personaggio contiene box, armatura e piano che contiene le istruzioni, mentre per la camera contiene la camera stessa e il piano delle istruzioni. Funzione che gestisce il cambio di personaggio controllato: dopo aver trovato quale è attivo, lo disattiva così come la sua camera e il relativo piano (che viene nascosto), e attiva il successivo e relativa camera con piano. Inoltre salva un'informazione di simultaneità (tramite la funzione *make_caption()*, che verrà analizzata in seguito) se il cambio avviene mentre un personaggio sta effettuando un'azione il quale, in questo caso, viene "congelato" nella corrente posizione.

reset(arms) arms: array di armature. Tre game properties presenti nelle armature, per evitare sovrascrizioni e conseguenti errori tra una scena e l'altra, hanno bisogno di essere azzerate ogni qual volta venga salvata una scena. Queste sono, e saranno analizzate più avanti, *simul*, *contemp* e *count*.

standPosition(arms, props) arms: array di armature, props: array di string contenente i nomi delle game properties di tutte le arms che se hanno valore True non applicano questa funzione. Funzione si occupa di rimettere i personaggi nella posizione base (utile in caso di azioni simultanee tra personaggi, per togliere la posizione congelata), da utilizzare quando si salva una nuova scena. Questo avviene solo quando nella singola armatura nessuna delle

props, se presente, ha valore True in quanto, altrimenti, alcune action risulterebbero non svolte. Ad esempio, se il cane è seduto avrà, e si vedrà più avanti il motivo, una game property "Sit" con valore True. Chiamando questa funzione nel momento in cui si salva una scena, si vedrebbe il cane tornare in piedi, e non avrebbe senso per il prosieguo della storia.

towards(obj1, obj2, A) obj1: personaggio che effettua l'azione, obj2: oggetto o personaggio verso il quale obj1 sta facendo l'azione, A: matrice di orientamento di obj1. Questa funzione calcola se obj1 sta effettuando un'azione verso obj2, dando in output True o False. Utile per verificare se il personaggio è direzionato verso il centro di un oggetto piano (come il suolo) che non è intercettabile con *rayCastTo()*.

listproperties(list, own) list: lista di nomi di alcune proprietà, own: possessore delle proprietà. Qui viene salvata nell'objectStorage la lista data.

storeProperties(base, propobjs) base: oggetto nella scena che possiede la game property *counter*, propobjs: oggetti con proprietà di cui salvare il valore (armatura + camera). Sono salvati, nell'objectStorage, i valori delle proprietà salvate con la funzione *listproperties()*.

storePosition(base, objs) base: oggetto nella scena che possiede la game property *counter*, objs: lista di oggetti che possono cambiare posizione o ruotare (box per i personaggi + camera). Tiene traccia, salvando nell'objectStorage di posizione e orientamento degli oggetti in ogni inquadratura salvata, così da poterle recuperare in caso se ne voglia rifare una.

restorePosition_and_Caption(base, objs, arms) base: oggetto nella scena che possiede la game property *counter*, objs: lista di oggetti che possono cambiare posizione o ruotare (box per i personaggi + camera), arms: lista di armature + oggetti che compiono azioni. Funzione che cancella i dati salvati relativi all'ultima scena, e porta gli objs nella precedente posizione. Grazie alle informazioni salvate tramite le game properties delle armature, il sistema riproduce anche una parte della action che si svolgeva nel precedente shot. Alla fine di essa, se non è vuoto, viene eliminato l'ultimo elemento dell'array captions tramite *pop()*.

make_caption(type, string, base, char) type: tipo di informazione che si vuole salvare, string: valore dell'informazione, base: oggetto nella scena che possiede la game property *counter*, char: oggetto da cui proviene l'informazione. Salva nel captionStorage tutte le informazioni utili per la generazione della descrizione che accompagnerà l'immagine. Il parametro type è fondamentale in quanto è quello cercato dal sistema nel momento in cui si compone

la frase. Si rende necessario, quindi, definirli con dei nomi che il sistema è in grado di comprendere (a meno di cambiarli in tutte le funzioni). Questi sono (facendo gli esempi sul personaggio "Dog"):

"Subject" con string di valore "the dog";

"Verb" per i verbi legati alle azioni compiute dall'armatura ("walks", necessario salvarli alla terza persona singolare inglese);

"Complement+verbo" perché sia distinguibile il verbo a cui è riferito, questo viene specificato nel nome ("Complementwalks" con string "in the house");

"Adverb+verbo" come Complement;

"Pos" è l'informazione di posizione legata al precedente verbo ("near the house");

"Verb_Pos+*count* -1" per i verbi legati alle informazioni di posizione e quindi rilevati nel box del personaggio (possono essere "is" e "has", anche qui, terza persona singolare). Da registrare dopo "Pos".

Vengono poi salvate altre informazioni necessarie:

"Timer" l'istante in cui l'azione viene effettuata;

"Contemp" è un boolean che indica se nel momento in cui un personaggio effettua un'azione, ne effettua anche un'altra (cane che cammina e abbaia);

"Simul" è un altro boolean ma indica se due o più personaggi effettuano un'azione nello stesso momento;

Se il type è "Verb" o "Pos", la chiave con cui viene salvata nel captionStorage l'informazione è nominata "type + valore di *count* + numero di scena -1 + nome del personaggio" e dopo averla registrata viene incrementato il valore di *count*. In questi due casi sono necessarie due cifre in quanto possono essere più di uno nella stessa scena, mentre per gli altri type la chiave è data da "type + numero di scena -1 + nome del personaggio".

change_words(key, i) key: termine da cercare se presente in vocab_db (spiegato nel prossimo punto), i: indice del sinonimo nell'array. Prende tutte le ripetizioni cercate e le sostituisce con un termine temporaneo uguale per tutte, poi le sostituisce uno ad uno con i sinonimi presenti. Dà in output il termine sostitutivo.

capitalize(text) text: stringa su cui effettuare l'operazione. Funzione che ha il compito di rendere maiuscolo il primo carattere dopo una frase, delimitata da ".!?" e che ha come output la stringa modificata.

showText(textobj, str) textobj: oggetto di tipo testo nel quale scrivere la didascalia della scena, str: stringa contenente la didascalia completa (si utilizza l'ultimo elemento presente in captions). Ha il compito di suddividere la stringa ogni 50 caratteri, trovare gli indici di tutti gli spazi presenti e inserire un "a capo" al posto dello spazio tra due parole più vicino. Inoltre dopo aver azzerato il delay sensor denominato *delay*, ne stabilisce un valore proporzionale al numero delle parole. Infine, assegna la nuova stringa al campo text del textobj.

print_caption(base, arms, objs) base: oggetto nella scena che possiede la game property *counter*, arms: lista di armature + oggetti che compiono azioni, objs: box dei personaggi. Funzione che si occupa dell'effettiva creazione della descrizione, che viene composta utilizzando tutte le informazioni salvate nel captionStorage. Ogni frase viene generata con la struttura soggetto + verbo + avverbio (se presente) + complemento (se presente). Si cerca poi nella frase costruita eventuali ripetizioni di termini presenti nel vocab_db, e che hanno quindi un sinonimo registrato, e ne fa una sostituzione tramite la funzione *change_words()*. Infine si utilizza la funzione *capitalize()* passandogli come parametro la stringa creata, tramite *append()* si aggiunge la stringa a captions e si genera come output la stringa modificata.

4.3.2 vocab.py

Breve menzione per un altro file creato a parte rispetto alla libreria. In questo breve script è stato inizializzato uno storage come *vocab_db =* all'interno del quale sono state create delle coppie chiave-valore dove la chiave è il termine destinato alla descrizione della scena utilizzato ripetutamente negli altri script o in *print_caption()*, mentre il valore è un array contenente tutti i sinonimi, nel quale il primo elemento è uguale al termine che definisce la chiave. Quest'ultimo passaggio è fondamentale per il funzionamento di *change_words()*.

Data l'estrema semplicità della struttura, questo piccolo dizionario è facilmente espandibile.

4.3.3 Personaggi (box)

Il box esterno dei personaggi necessita di alcuni sensori, di uno script come controllore e di alcuni attuatori, oltre che di qualche game properties.

game properties *active*, boolean che diventa "True" quando il personaggio è controllato, *coll*, usata per essere rilevato come oggetto con cui collidere (non importa il type), *character*, utile per i sensori di relazione tra personaggi (non importa il type) e *count*, integer che tiene il conto di quante informazioni di

posizione sono registrate di modo che poi il sistema sappia quante cercarne nel momento in cui deve creare la descrizione testuale. Se il personaggio dovrà essere intercettato dalla direzione di un altro, avrà anche una game property caratteristica (qui utilizzato il nome del box stesso in minuscolo: per la ragazza *girl*, per il cane *dog* e per la pecora *sheep*). Ci possono poi essere altre game properties relative all'utilizzo del personaggio in una particolare scena.

keyboard sensors Qui ve ne sono quattro (tasti W, A, S, D) che hanno il compito di attivare gli attuatori legati al movimento.

collision sensor Viene specificata una game property (qui denominata *coll*) e, se attivato, provoca lo stop del movimento che avviene tramite script.

radar sensor Si definisce la game property (qui *character*) e viene utilizzato per definire se un personaggio ne sta guardando un altro: è stato infatti, e deve essere, chiamato *look*.

near sensor Anche in questo caso si definisce la game property (*character*). Usato per specificare quali personaggi sono vicini tra loro: anche in questo caso, il nome deve essere *near*.

ray sensors Ve ne sono cinque, uscenti dalle facce del parallelepipedo (escludendo quella verso il basso) e intercettano una determinata game property (*coll*). Utilizzati per delineare la posizione del personaggio in relazione agli oggetti di contesto.

always sensor Anche in questo caso va attivato il *TRUE level triggering*, così che possano essere continuamente aggiornate le informazioni legate alle due camere presenti sul personaggio e fare dunque il cambio quando necessario. Se nel personaggio un input porta a più di una action (ad esempio, il cane si alza se seduto e inizia a camminare con lo stesso comando) nei relativi sensori è necessario spuntare la casella *Tap*.

controller Uno script gestisce le informazioni ricevute dai precedenti sensori, e se ne può definire una struttura standard. Prima cosa da fare è importare le librerie necessarie, compresa quella precedentemente realizzata. Dopodiché, grazie alle funzioni (precedentemente citate) della libreria bge, si ottengono gli oggetti della scena necessari, l'oggetto al quale sono assegnati i logic bricks, i sensori e gli attuatori. Dopo aver azzerato la property *count*, si definiscono le azioni da fare se la property *active* ha valore "True". Questo consiste nel controllare se un sensore è stato attivato e, in caso positivo, attivare un attuatore passandogli i valori necessari per muoversi o per ruotare su sé stesso. Dopo il controllo dei sensori, si verifica se il ray sensor posseduto dalla

camera con ampia inquadratura è positivo. In tal caso si rende attiva l'altra camera e viceversa. Tutta la parte seguente, va realizzata a prescindere dal valore di *active*. In caso di positività del collision sensor, si blocca il personaggio passando il valore 0 a tutti i parametri dell'attuatore corrispondente al movimento. Si calcola poi il numero di ray sensors attivi, specificando di non contare quando l'oggetto intercettato ha anche la game property *character*. Da qui in avanti si utilizza, se si verificano determinate condizioni, la funzione *make_caption()*. La si utilizza prima per registrare un "Subject", poi, tramite l'attivazione di determinati sensori, si registra un "Pos" e un "Verb_Pos+*count*-1". Importante inserire, per ogni *if* che controlla se uno o più sensori sono attivi e di conseguenza salvano informazioni, un else che controlla se in captionStorage è presente un "Pos" come quello definito nell'*if* e, in caso positivo, lo cancelli insieme al relativo "Verb_Pos". Questo passaggio serve ad aggiornare continuamente l'effettiva informazione di posizione senza accumularne diverse nel captionStorage. Ad esempio, se un personaggio inizialmente si trova vicino ad un punto di interesse, verrà salvata la relativa informazione. Se però si deve spostare e nel momento in cui viene catturata l'immagine non si trova più vicino al punto di prima, deve essere eliminata l'informazione precedentemente salvata e aggiunta una nuova, se presente. Questo è possibile in quanto l'ultima parte è scritta al di fuori del controllo sul valore di *active* e quindi questi sensori vengono controllati continuamente. Nel dettaglio, per i ray sensors, se se ne attivano almeno due incluso quello che punta verso l'alto, viene definita l'informazione di trovarsi all'interno di qualcosa, mentre ne basta uno solo, che non sia quello verso l'alto, per stabilire la vicinanza.

motion actuators Sono inoltre presenti tre motion actuators che nei logic bricks hanno tutti i valori azzerati, in quanto vengono poi assegnati nel codice per una maggiore elasticità. Questi, come visto nel controllore, vengono attivati se prima si ha un output positivo del relativo sensore.

Per quanto riguarda la fisica, a questo oggetto viene assegnato il tipo *character* e spuntata la casella *actor* che permette al box di essere rilevato dai sensori di tipo near e radar. Inoltre è reso invisibile alla renderizzazione in atto quando si avvia il BGE ed ha *collison bounds* attivi aventi *Convex Hull* come *shape*.

Tabella 4.1: Riassunto elementi base box

Nome	Logic element	Tipo	Nome obbligatorio
active	game property	Boolean	Si
coll	game property	Boolean	Si*
character	game property	Boolean	Si*
count	game property	Integer	Si
w	sensor	Keyboard	No
a	sensor	Keyboard	No
s	sensor	Keyboard	No
d	sensor	Keyboard	No
collision	sensor	Collision	No
near	sensor	Near	Si
look	sensor	Radar	Si
ray_f	sensor	Ray	No
ray_b	sensor	Ray	No
ray_r	sensor	Ray	No
ray_l	sensor	Ray	No
ray_u	sensor	Ray	No
always	sensor	Always	No
python	controller	Python	No
move	actuator	Motion	No
right	actuator	Motion	No
left	actuator	Motion	No

*i sensori di rintracciamento cercheranno questa game property, basta che il nome corrisponda

4.3.4 Personaggi (armatura)

Come per l'altra componente sono presenti game properties, sensori e un controllore di tipo script, mentre non sono necessari attuatori.

game properties Ce ne sono di due tipi: fisse e variabili. Quelle fisse, che quindi devono avere tutte le armature, sono *count*: un integer che conta quante azioni differenti vengono svolte in ogni scena, *simul*: boolean al quale il sistema assegna True se l'azione avviene in contemporanea con quella di un altro personaggio, e *contemp*: altro boolean che viene posto True se il personaggio effettua più azioni insieme. Quelle variabili, invece, sono legate alle action delle quali è utile avere memoria se esse stavano avvenendo o meno nel momento in cui è stata definita un'inquadratura. Perché il sistema le riconosca, e nel dettaglio la funzione *restorePosition_and_Caption()*, devono avere

lo stesso nome dell'action a cui fanno riferimento (senza nome dell'armatura, quindi la game property legata alla camminata si chiamerà solo *Walk*). Sono dei boolean e sono dunque variabili in quanto ogni personaggio avrà le proprie in base alle action che può compiere. Tra queste, l'unica che posseggono tutti è quella relativa alla posizione base da assumere subito dopo che viene confermata un'inquadratura (qui definita *Stand*). Anche in questo caso ci possono essere altre game properties utili allo sviluppo della scena.

keyboard sensors In questo caso il numero è variabile in base a quanti tasti vengono destinati all'attivazione di action. In genere vi sarà lo stesso del box (in questo caso W) che attiva l'azione di camminata e un altro (S) che la porta a termine.

always sensor Utilizzato per permettere al sistema di salvare alcune informazioni specificate nello script, anche se la specifica armatura non è stata attivata e per aggiornare continuamente le informazioni relative ai marker e alle frecce (bisogna spuntare la casella del *TRUE level triggering*). Come visto nel box, bisogna spuntare *Tap* in alcuni, se presenti, sensori.

controller Come per il box, il controllore è di tipo Python e lo script definito può essere ricondotto ad una struttura standard. La prima parte è uguale: si importano le librerie necessarie e si creano variabili contenenti gli oggetti e i sensori da usare in seguito. Anche in questo caso, c'è una parte da compilare solo quando la game property *active* del box/padre ha valore True. Le prime due cose da fare sono salvare in una variabile ("A") la matrice di orientamento dell'armatura tramite la funzione *worldOrientation* (libreria bge) e registrare il "Subject" tramite *make_caption()*. Dopodiché, all'attivazione di un sensore (e, se necessario, con un determinato valore di una variabile o game property) bisogna riprodurre la action tramite la funzione *playAction()* (libreria bge) e salvare determinate informazioni. In caso sia una action che può avvenire in contemporanea con un'altra, si controlla se ce n'è già una in riproduzione e si pone di conseguenza il valore di *contemp*. Si utilizza poi per tre volte la funzione *make_caption()* salvando informazioni di tipo "Timer", "Contemp" e "Verb" e si conclude ponendo, se necessario, a True o False le varie game properties legate alle action. Dopo aver fatto questo passaggio per tutte le action da riprodurre, si possono aggiungere dei complementi ai verbi appena salvati. Anche in questo caso, al verificarsi di determinate condizioni, si utilizza *make_caption()* con type "Complement+verbo" (ad esempio, dopo aver verificato che l'action in svolgimento è quella di camminata e che *rayCastTo()*, che utilizza l'empty per calcolare la direzione, una distanza a piacere e una stringa che rappresenta una game property che l'oggettto da intercettare deve

possedere (in questo caso *girl*), ha valore True, si può registrare un "Complementwalks" con string "towards the girl"). Per i complementi, così come per gli avverbi, non c'è un'indicazione fissa, basta che vengano registrati al verificarsi delle condizioni che li rappresentano. Si rieffettuano poi questi controlli per rendere visibile la freccia verde o rossa in base a se si è direzionati verso un punto di interesse o meno. Se al personaggio sono legati altri marker, si effettua il controllo anche sulla loro visibilità. Quando *active* ha valore False, si rendono invisibili entrambe le frecce e evnetuali altri marker che solo quel personaggio può visualizzare. Uscendo dal controllo di *active*, si assegna un valore a *Stand*. Questo è True se tutte le altre game properties variabili sono False, altrimenti è False. Infine si definisce un array composto dai nomi delle game properties variabili, e lo si passa come primo parametro alla funzione *listproperties()*.

Tabella 4.2: Riassunto elementi base armatura

Nome	Logic element	Tipo	Nome obbligatorio
simul	game property	Boolean	Si
contemp	game property	Boolean	Si
count	game property	Integer	Si
w	sensor	Keyboard	No
s	sensor	Keyboard	No
always	sensor	Always	No
python	controller	Python	No

Ha come tipo di fisica *No collision*, in quanto le collisioni sono gestite dal box/padre, ed è importante che il suo orientamento locale sia lo stesso del padre e che la sua scala sia 1.0 in tutte e tre le dimensioni.

4.3.5 Personaggi (altre componenti)

Delle quattro componenti aggiuntive precedentemente introdotte, solo la camera "in prima persona" non necessita di alcun elemento aggiuntivo.

La camera con ampia visuale, possiede un ray sensor con la game property *coll*, lungo l'asse Z negativo e con una lunghezza che deve essere minore della distanza tra questa camera e il box del personaggio. Quando questo sensore risulta positivo, verrà attivata l'altra camera, come visto nello script controller del box.

L'empty padre, invece, deve possedere una game property chiamata *active*, sempre di tipo boolean, un mouse sensor, che rileva, in questo caso, il movimento, un property sensor, specificando la game property *active* e il valore True, un AND controller e un mouse actuator in modalità *look*.

Il piano che mostra le istruzioni ha la struttura più complessa. Ha due game properties di tipo boolean, chiamate *active* e *visibility*, delle quali quest'ultima va posta a True spuntando la corrispondente casella. Vi sono poi tre property sensors, di cui due con *visibility* (uno True e uno False) e uno con *active* (True), e un mouse sensor con evento il click del tasto sinistro (che mostrerà/nasconderà l'oggetto). Sono gestiti da due AND controllers che hanno tre input ciascuno: mouse sensor, property sensor (*active*) e uno degli altri due property sensors (*visibility*). Questi due controllori hanno due output ciascuno che portano a un totale di quattro attuatori: due property actuators, che mettono alla game property specificata il valore specificato, e due visibility actuators, qui utilizzati per rendere visibile o invisibile l'oggetto. L'AND che aveva in input il sensore con *visibility* True, ha in output la coppia di attuatori che rendono *visibility* False e invisibile l'oggetto, e viceversa. Inoltre è reso invisibile alla renderizzazione così che non si veda quando si controlla la camera, mentre lo si possa visualizzare, grazie a logic bricks e codice, quando si controlla il personaggio. Il tipo di fisica di questi tre oggetti è *No Collision*.

4.3.6 Gruppo di personaggi

I singoli personaggi facenti parte di un gruppo, subiscono alcune modifiche rispetto a quanto visto in precedenza. Le differenze qui descritte sono legate alla struttura precedentemente descritta.

box Rimane solo la game property *active* più eventuali altre che possono variare nei casi. Per quanto riguarda i sensori, si tolgono tutti tranne quelli di tipo keyboard e il collision, e se ne aggiunge uno di tipo always. Nello script controller, oltre ad assegnare all'empty1 la posizione del centro di massa del gruppo, l'unica differenza è l'aggiunta di un azzeramento del motion actuator quando *active* ha valore False. Questo nei personaggi avviene in automatico tramite la funzione *controlled()*. Avendo una struttura diversa, per i gruppi è necessario specificarlo in questo modo.

armatura Si toglie la game property *count*, e si aggiunge un always sensor. Come per il box, si deve aggiungere nello script uno stop all'action in riproduzione, se ce n'è una, quando *active* del box è False.

Come visto al punto 4.2.2, ci sono poi due empty, uno autonomo (empty1) e un altro (empty2) che è padre di tutti i box del gruppo.

empty1 Ha tre game properties: *active*, *character* e *count*. Le ultime due sono quelle che il box non possiede più. Dal box ha preso anche alcuni sensori, come quelli di tipo ray e near: essendo ora legati a questo empty che ha come posizione il centro del gruppo, i valori di questi sensori vanno aumentati. Per

i ray sensors, inoltre, è necessario attivare la *X-Ray Mode* tramite l'apposita casella (come visto, i box non possiedono più *coll*). Il controllore a cui sono legati è di tipo AND.

empty2 Possiede solamente due game properties: *active* e *count*, quest'ultima sostituisce quella che non possiedono più le armature.

4.3.7 Macchina da presa

Anche per quanto riguarda la camera principale, la fase di sviluppo richiede diversi elementi.

game properties Ne sono necessarie due: *active*, boolean con valore iniziale True (spuntando la casella) e *field*, di tipo string.

keyboard sensors Quattro tasti necessari, che verranno utilizzati dallo script per attivare il movimento della camera (anche in questo caso, W, A, S, D). Inoltre, ne è stato definito un quinto per permettere il cambio di lunghezza focale in tempo reale.

mouse sensor Utilizza *Movement* come tipo di evento.

radar sensors Tre sensori che si attivano tutti con la stessa game property *active*, con lo stesso angolo e sullo stesso asse (-z), ma con tre valori di distanza diversi.

collison sensor Anche in questo caso la game property specificata è *coll*, serve a bloccare la camera in caso di collisioni.

always sensor Anche qui utilizzato per salvare determinate informazioni a priori e per le informazioni relative ai marker, e anche in questo caso si spunta *TRUE level triggering*.

controller Il controllore è anche in questo caso uno script e la sua struttura segue praticamente lo stesso schema dei precedenti. Dopo aver importato le librerie necessarie, si creano variabili contenenti oggetti, sensori e attuatori da usare in seguito. Tra questi, viene definito un array contenente valori assegnabili al campo relativo alla lunghezza focale. Si controlla quindi che *active* abbia valore True, e in tal caso si attivano: gli attuatori del movimento allo stesso modo fatto per i box dei personaggi, quello di rotazione legato al mouse, e si gestisce, tramite il sensore dedicato, il cambio di lunghezza focale che, oltre a questo compito, provvede anche a modificare la posizione dell'oggetto testo (oltre che renderlo visibile per qualche frame) e del piano contenente le

istruzioni, così che siano sempre visibili. Tramite *setVisible(False)* poi, si nascondono tutti gli oggetti che non si vogliono mostrare nel momento in cui si controlla la camera (come ad esempio i marker posti per terra per delimitare o indicare qualcosa) così da non averli anche nell'immagine che comporrà lo storyboard. Quando *active* ha valore False, si gestiscono eventuali visualizzazioni degli oggetti che prima sono stati nascosti. Al di fuori del valore di *active*, si controllano i tre radar sensors e, in base a quale è attivo, si assegna a *field* il valore "Close up", "Cow-boy shot" o "Wide shot". Qui si effettua anche il controllo del collision sensor. Come nel caso delle armature, si crea un array con i nomi delle game properties (fatta eccezione per *active*, quindi in questo caso solo *field*) e lo si registra tramite *listproperties()*.

motion actuator Un solo attuatore di movimento, utilizzato per tutte le quattro direzioni.

mouse actuator Ad esso è associata la rotazione della camera.

Tabella 4.3: Riassunto elementi base camera

Nome	Logic element	Tipo	Nome obbligatorio
active	game property	Boolean	Si
field	game property	String	Si
w	sensor	Keyboard	No
a	sensor	Keyboard	No
s	sensor	Keyboard	No
d	sensor	Keyboard	No
lens	sensor	Keyboard	No
collision	sensor	Collision	No
radar1	sensor	Radar	No
radar2	sensor	Radar	No
radar3	sensor	Radar	No
always	sensor	Always	No
python	controller	Python	No
move	actuator	Motion	No
mouse	actuator	Mouse	No

Ha la fisica di tipo *Dynamic*, con le caselle *Actor* e *Ghost* spuntate, di modo da poter rilevare le collisioni in modo corretto e allo stesso tempo non essere un ostacolo che blocca il movimento dei personaggi. Possiede *Cone* come *collision bounds* ed è necessario spuntare anche le tre caselle relative ai tre assi in *lock translation*, di modo che la camera non subisca gli effetti della gravità e si muova solo tramite i sensori dedicati.

4.3.8 Macchina da presa (altre componenti)

Delle altre due componenti, ovvero il piano per le istruzioni e l'oggetto testo, la prima ha le stesse caratteristiche dei piani per le camere dei personaggi visti in precedenza, con le sole differenze che *active* ha valore True di default e che è inizialmente reso visibile alla renderizzazione.

L'oggetto testo ha quattro keyboard sensors, con i due tasti utilizzati per salvare e cancellare un'inquadratura, quello usato per cambiare la focale della camera e quello che cambia personaggio e un delay sensor (da nominare *delay* in quanto utilizzato in *showText()*)

La gestione è affidata a tre controllori (un OR e due AND): l'OR ha in input i keyboard sensors tranne quello del cambio personaggio, un AND ha in input il delay sensor e l'altro AND ha in input il restante keyboard sensor.

Ogni controllore ha in output un attuatore: dall'OR si arriva ad un visibility actuator che rende l'oggetto visibile, mentre dai due AND si giunge ad un altro visibility actuator con valori ed effetti contrari.

Ha fisica di tipo *No Collision* ed è reso inizialmente invisibile.

4.3.9 Oggetto "Ground"

Alcune azioni più generali, e non relative ad un singolo personaggio, sono state assegnate ad un oggetto esterno per mantenere più ordinato il codice. L'oggetto scelto è stato, in tutti i casi d'uso, quello utilizzato come suolo o parte di esso, da qui il nome "Ground". Non è comunque necessario che sia quello l'oggetto da utilizzare per i compiti spiegati di seguito.

game properties Un integer di nome *counter*, che tiene il conto delle scene salvate e un timer di nome *time*, utilizzato come valore del campo string nella funzione *make_caption()* ogni qual volta si registri un "Timer". Solo la prima necessita quel nome specifico.

keyboard sensors Sono quattro, destinati a: cambio del personaggio (qui tasto Space), salvataggio inquadratura (Enter), cancellazione ultima inquadratura salvata (Canc) e uscita dal gioco con apertura automatica dello storyboard realizzato (Esc).

always sensor Utilizzato per far sì che lo script controller venga compilato appena avviato il BGE.

controller La prima parte è come le precedenti, con l'importazione delle librerie e la creazione di variabili contenenti sensori e oggetti utili. Dopodiché si fanno alcuni controlli. Quando il valore di *counter* è 0, ovvero all'avvio del

BGE, si effettuano alcune operazioni: si rende attiva la camera principale tramite *active_camera* (libreria bge), si pulisce il captionStorage con la funzione di Python *clear()*, si salva in una variabile il *path* relativo al file Blender su cui si sta lavorando tramite *bpy.path.abspath("//")* e lo si rende assoluto, così che il file html e le immagini prodotte siano nella stessa cartella del file Blender, tramite *chdir()*. Ci sono poi tre operazioni legate all'oggetto testo, ovvero si imposta la risoluzione, in quanto di default ha un valore molto basso, e, dopo avergli assegnato la stessa *worldPosition* della camera, si assegna alla sua *localPosition.z* un valore pari a -focale/17. Dopodiché si crea il file per lo storyboard in tre passaggi: si apre un file html in modalità "writing" (cioè che sovrascrive ciò che già era presente) tramite la funzione Python *open()* e lo si assegna ad una variabile, da questa variabile si utilizza la funzione *write()* all'interno della quale si inizializza il file tramite codice html (scrivendo per riga, ognuna di esse va messa tra apici singoli), infine si chiude il file tramite *close()* (Appendice A.1). Come ultimo passaggio si pone *counter* a 1. Se il sensore destinato al cambio personaggio risulta positivo, si utilizza la funzione *controlled()* passando tutti i parametri necessari. Quando viene attivato il sensore dedicato al salvataggio della scena e in contemporanea l'oggetto attivo è la camera, bisogna fare diverse operazioni. La prima è salvare l'immagine tramite *makeScreenshot()* e per far sì che venga salvata e poi letta correttamente, bisogna utilizzare *sleep()* che permette a Blender, anche con un tempo minimo, di registrare tutto correttamente. Dopodiché si riapre il file html inizializzato in precedenza, sempre utilizzando *open()*, in modalità "append" (che aggiunge in coda), si crea una variabile contenente l'immagine creata in un formato leggibile in html e si inizializza un'altra variabile che contiene il codice html di creazione di una card alla quale si assegna come ID il numero della scena rappresentata, si mettono in alto i dettagli della camera, l'immagine al centro, un titolo scritto direttamente in queste linee contenente il luogo e il dettaglio "interno/esterno" e infine la descrizione ottenuta tramite *print_caption()*. Tramite *write()* si scrive la card e con *close()* si chiude il file (Appendice A.2). Si invocano poi alcune delle funzioni realizzate, nell'ordine: *showText()*, *storeProperties()*, *storePosition()*, *reset()* e *standPosition()*. Dopodichè si controlla il sensore che cancella la precedente inquadratura, nel quale si chiamano *showText()* (come da descrizione se captions contiene almeno un elemento, con una stringa vuota nel secondo parametro altrimenti) e *restorePosition_and_Caption()*. Successivamente si riapre il file html in modalità "reading" (sola lettura) e si assegna ad una variabile l'intero contenuto del file in formato string. Si cerca nella stringa, tramite *find()* e grazie all'ID, l'ultima card salvata e si assegna ad una nuova variabile la parte di stringa trovata. Dalla stringa iniziale si cancella tutto ciò che si trova dopo la seconda stringa estrapolata, di modo quindi da eliminare la suddetta card. Si

chiude il file, e lo si riapre in modalità "writing", vi si scrive la nuova stringa modificata senza l'ultima card e lo si chiude per l'ultima volta (Appendice A.3). L'ultimo controllo è relativo al sensore di uscita, dove si riapre il file in "append" e vi si scrive, tramite *write()*, una stringa contenente il valore della game property *time* che indica il momento di uscita dal gioco, nonché il tempo impiegato. Dopo averlo chiuso si utilizza *open_new()* passandogli il nome del file html creato, per poi chiudere il gioco con *endGame()*.

Tabella 4.4: Riassunto elementi base camera

Nome	Logic element	Tipo	Nome obbligatorio
counter	game property	Integer	Si
time	game property	Time	No
change	sensor	Keyboard	No
screen	sensor	Keyboard	No
back	sensor	Keyboard	No
exit	sensor	Keyboard	No
always	sensor	Always	No
python	controller	Python	No

La fisica di questo oggetto dipende dalla sua natura. Nei casi d'uso, avendo utilizzato il suolo, questo ha il tipo *Static* con *collision bounds* attivi di forma *Box*.

4.3.10 Altri oggetti

Tutti gli altri oggetti in scena, se non hanno motivo di avere particolari caratteristiche, hanno la fisica *Static* con *collision bounds* di tipo *Box* o *Triangle Mesh* in base all'oggetto.

Inoltre, tutti quelli che potrebbero trovarsi a contatto con un personaggio, devono avere la game property *coll* di modo da essere riconosciuti in caso di collisione, e un'altra game property (in genere il nome dell'empty padre) di modo da essere riconosciuto da *rayCastTo* quando questa viene utilizzata per specificare la direzione del personaggio.

Se invece un oggetto compie un'azione, come nel primo caso d'uso per la gabbia che si apre, questo dovrà avere uno script controller che salva le informazioni generate, sempre se queste siano da inserire nella descrizione della scena. Si avrà quindi il controllo di un sensore che, se attivo, permetterà di salvare le informazioni precedentemente viste ("Timer", "Contemp" e "Verb") oltre che un "Subject", che viene inserito all'interno del controllo così da non salvarlo ad ogni scena ma solo quando viene utilizzato.

4.3.11 Eccezioni

Come accennato ad inizio capitolo, in base al tipo di scena o a ciò che si vuole raccontare, potrebbero essere necessari degli elementi aggiuntivi.

Riprendendo dal punto 4.3.10, nel terzo caso d'uso c'è un oggetto che ha un compito simile a quello della gabbia, con la differenza che salva informazioni come se le stesse compiendo il cane: bisogna quindi fare attenzione ai parametri da passare a *make_caption()*.

Un altro esempio, più semplice, lo si trova nel secondo caso d'uso, nel quale l'armatura del cane possiede alcuni sensori in più in quanto dovrà riprodurre più action. Di conseguenza anche il relativo script dovrà avere qualche linea di codice in più per contemplare anche i controlli sui sensori aggiuntivi.

Qualcosa di più complesso lo si trova nel terzo caso d'uso, nel quale sia il cane che la pecora presenti nella scena, subiscono una "trasformazione". Per tradurre questa idea nel BGE, è stato necessario aggiungere degli elementi agli oggetti mesh dei personaggi in questione, in quanto è richiesta una modifica alla mesh stessa. Nel dettaglio, oltre ad una game property utile nel momento in cui avviene il cambiamento, è utilizzato un edit object actuator che effettua lo scambio con un'altra mesh precedentemente realizzata. Queste due sostituzioni avvengono in modi diversi tra loro, in quanto una si attiva nel momento in cui la pecora si avvicina ad un determinato oggetto che a sua volta possiede una certa game property, mentre per il cane è l'oggetto stesso a riconoscere la vicinanza del personaggio e ad attivare un nuovo script che permette di attivare il cambio della mesh.

Questo per sottolineare come ogni situazione realizzabile possa aver bisogno di un qualunque elemento in più o in meno ma che introducendoli, si rispettino le indicazioni precedentemente illustrate, per far sì che il sistema funzioni correttamente.

4.4 Generazione delle frasi

Come visto nel sotto-capitolo 4.3.1, ci sono alcune funzioni realizzate per far sì che il sistema sia in grado di generare le frasi. Nel dettaglio, le due principali sono *make_caption()* e *print_caption()*. Tramite la prima si salvano tutte le informazioni necessarie, mentre la seconda si occupa di prendere queste informazioni e creare delle frasi. Assunto come queste vadano utilizzate, ora verrà analizzato nel dettaglio il lavoro effettuato da *print_caption()*. Ci sono tre possibili combinazioni di frasi tra personaggi:

- stessa azione effettuata nello stesso momento;
- azioni differenti effettuate nello stesso momento;

- azioni uguali o differenti effettuate in momenti diversi.

Queste vengono realizzate confrontando delle liste create sul momento che raccolgono verbi, avverbi, complementi e informazioni di contemporaneità. Se queste liste combaciano, i due personaggi in esame in quel momento stanno effettuando la stessa azione e viene salvata un'informazione di pluralità. L'informazione salvata in "Simul", dice anche se avvengono nello stesso momento e se essa ha valore True, i diversi soggetti vengono uniti e il verbo coniugato al plurale (figura 4.4a). Se, pur avendo "Simul" valore True, le liste non combaciano, il sistema scrive le due azioni interponendo tra loro un "meanwhile" (figura 4.4b). Quando, invece, ha valore False, significa che le due azioni non sono contemporanee, e quindi il sistema le scrive come sequenziali aggiungendo un "then" tra le due (figura 4.4c). Inoltre, se un personaggio è in grado di effettuare più azioni contemporaneamente, il sistema controlla l'informazione "Contemp" e le pone in sequenza dopo il soggetto: se ha valore True aggiunge un "and" tra i verbi, altrimenti inserisce "and later" nel caso in cui queste siano state effettuate in sequenza (figura 4.4d).

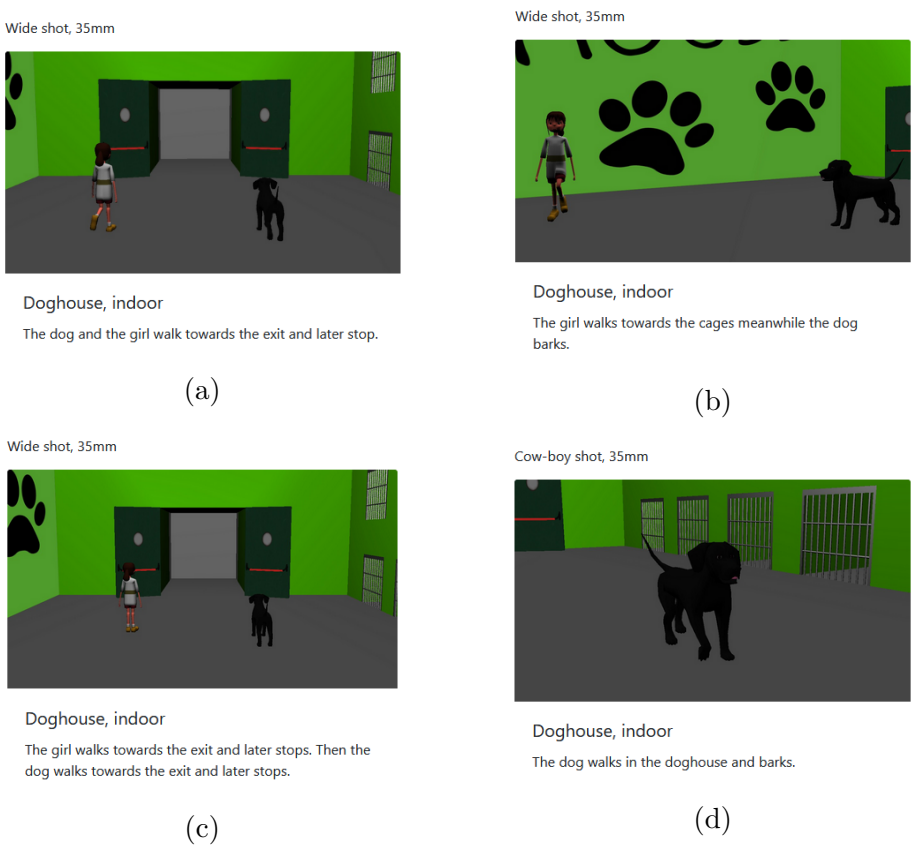


Figura 4.4: Esempi di frasi relative alle azioni

Allo stesso modo, per le informazioni di contesto, se più soggetti, per esempio, sono “near the house”, essi vengono uniti e il verbo coniugato al plurale (figura 4.5a). In questa parte, le informazioni relative a due o più personaggi vicini tra loro o che si guardano reciprocamente, sono gestite separatamente. Infatti, quando un personaggio ne guarda un altro, si salva come string (tramite *make_caption()*) “looking at the ” + il nome del personaggio intercettato dal sensore. Se il sistema rileva che la situazione è reciproca, modifica la frase unendo i soggetti e aggiungendo “are looking at each other” (figura 4.5b). Per la vicinanza tra personaggi, invece, il sistema utilizza il sensore dedicato, per creare liste di personaggi intercettati e, dopo aver eliminato i doppioni e eventuali sottoliste, unisce i soggetti trovati e li lega a “are near each other” (figura 4.5c). In modo simile alle azioni, se più soggetti condividono le stesse informazioni di contesto, il sistema genera un’unica frase unendo i soggetti (figura 4.5d).

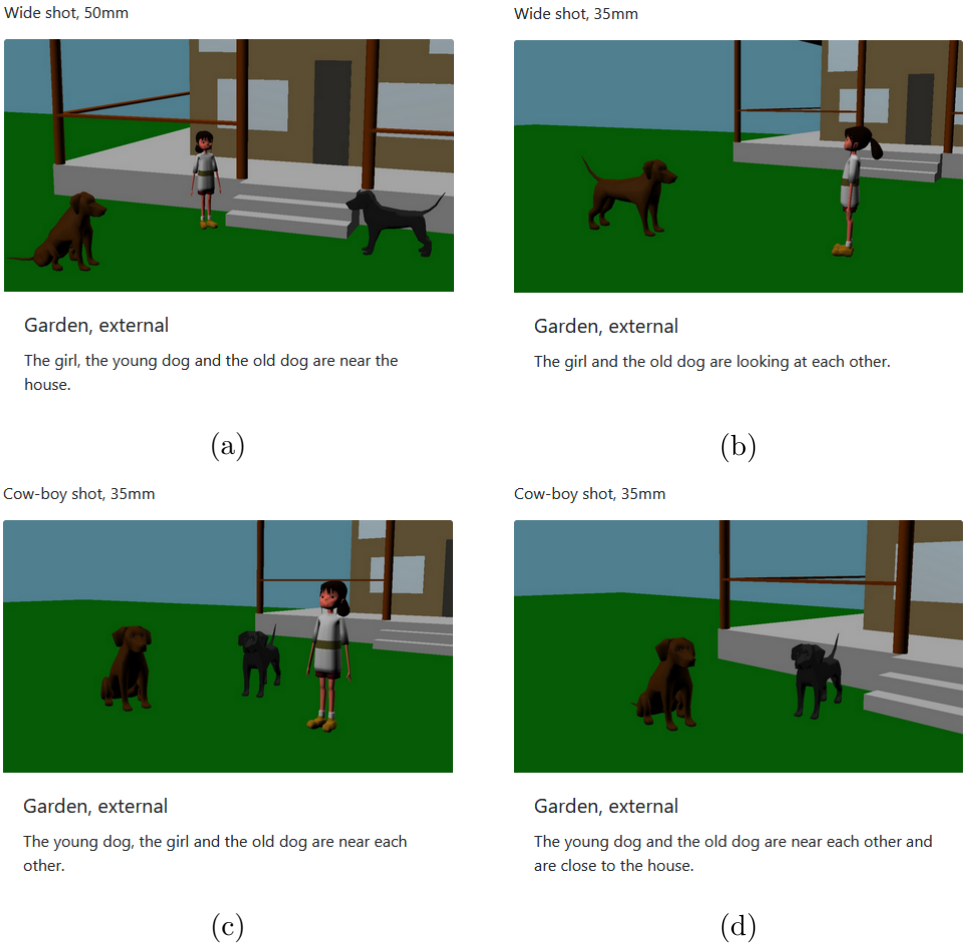


Figura 4.5: Esempi di frasi relative alle informazioni di contesto

Capitolo 5

Use cases e test

In questo capitolo si mette in pratica tutto ciò a cui le fasi di design e sviluppo hanno portato. Questo prevede, dunque, la realizzazione dei casi d'uso e i test nei quali sono degli utenti a doverne ricreare uno e dai quali trarre dei feedback.

5.1 Use cases

In una fase preliminare dello sviluppo del progetto, sono stati definiti tre casi d'uso sui quali applicare il sistema sviluppato. Essi sono quindi delle scene, delle quali verrà realizzato uno storyboard, che hanno un protagonista comune (un cane) e che, per l'ordine in cui sono pensate, seguono sia un ordine narrativo che un ordine di complessità.

5.1.1 Use case 1

La prima scena prevede la presenza di due personaggi (il cane e una ragazzina) i quali si trovano in un canile; la ragazzina sceglierà il cane e lo porterà via con sé.

In questo primo caso d'uso si può vedere già molto di cosa può fare il sistema. Ad esempio, come sia in grado di gestire la realizzazione di determinati tipi di frase, quali due personaggi che effettuano la stessa azione in contemporanea o sequenzialmente, così come un personaggio che effettua più azioni nella stessa inquadratura. Inoltre si può anche vedere come crei le frasi di contesto nel momento in cui i due personaggi sono vicini ad uno stesso punto, vicini tra loro o si stanno guardando reciprocamente. Un'altro dettaglio importante già introdotto qui, è il fatto che anche altri oggetti possono effettuare azioni da memorizzare nello storage. In questo caso, la gabbia che si apre salva delle informazioni come fosse un personaggio.

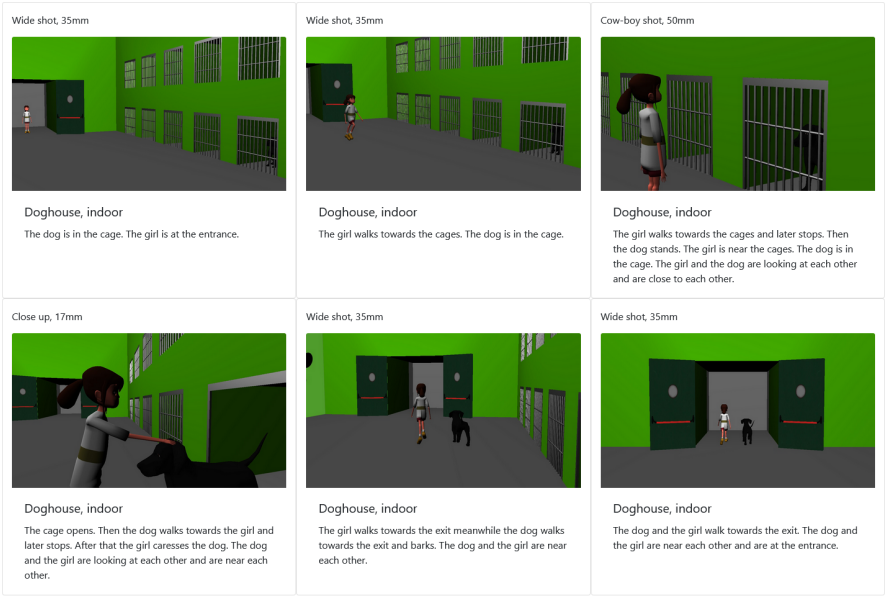


Figura 5.1: Storyboard del caso d'uso 1 realizzato con l'applicativo

5.1.2 Use case 2

Nella seconda scena vi sono gli stessi due personaggi della prima più un altro cane che la ragazzina già aveva, e l'ambientazione è il prato che circonda la casa in campagna della ragazzina; qui, verranno insegnati al cane nuovo dei comandi, anche con l'aiuto del cane più anziano.

Figura 5.2: Parte dello storyboard de "Lilo & Stitch" che ha ispirato questo caso d'uso [52]

Oltre alle caratteristiche visibili nel primo caso d'uso, qui viene aggiunto un terzo personaggio, che mostra come il sistema sia in grado di gestire un numero indefinito di personaggi, a patto che siano costruiti e passati alle funzioni in modo corretto. Un'altra aggiunta, anche se di minore importanza, è il maggior numero di azioni che i personaggi possono effettuare rispetto alla precedente scena, oltre che un controllo sui sensori dei personaggi effettuato in modo più pratico per il senso della storia (ad esempio, per far applaudire la ragazza non basta schiacciare il tasto corrispondente ma c'è bisogno che il cane abbia obbedito ad un comando, così come il comando non può essere dato se non da vicino e guardando il cane, e così come anche il cane non può eseguire un comando se prima la ragazza non glielo chiede).

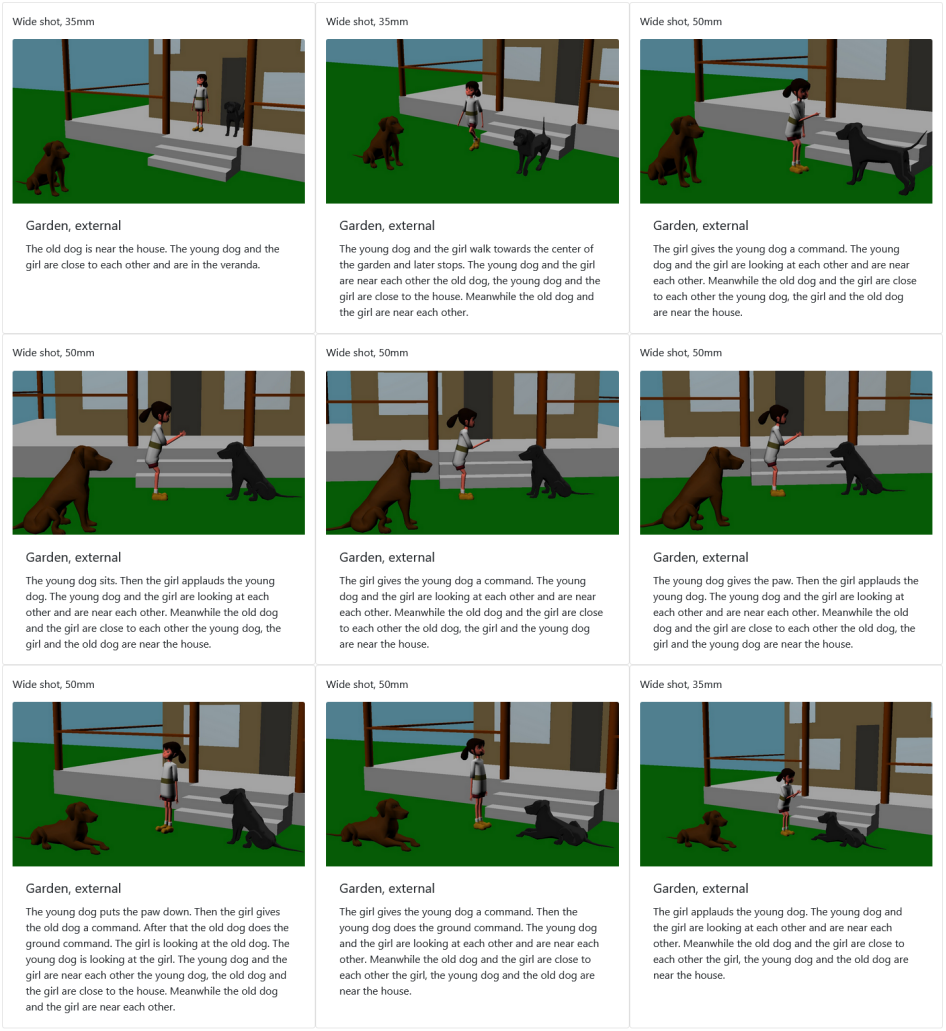


Figura 5.3: Storyboard del caso d'uso 2 realizzato con l'applicativo

5.1.3 Use case 3

La terza ed ultima scena prevede tre personaggi tra i quali non figura più la ragazzina, mentre è presente il cane, un gregge di pecore ed una pecora "ribelle" che si trovano al pascolo; il cane è ormai diventato un cane-pastore e controlla le pecore, tra di esse una si stacca dal gregge e per sbaglio entra nel capannone della tosatura; il cane, per inseguirla, cade in un piccolo fossato che era coperto da un mucchio di foglie raccolte; la pecora uscita dal capannone è ora tosata e snella mentre il cane uscito dal fossato è coperto da un manto di foglie; si invertono così i ruoli con la pecora che intima al cane di unirsi al gregge, il quale viene poi da lei controllato.

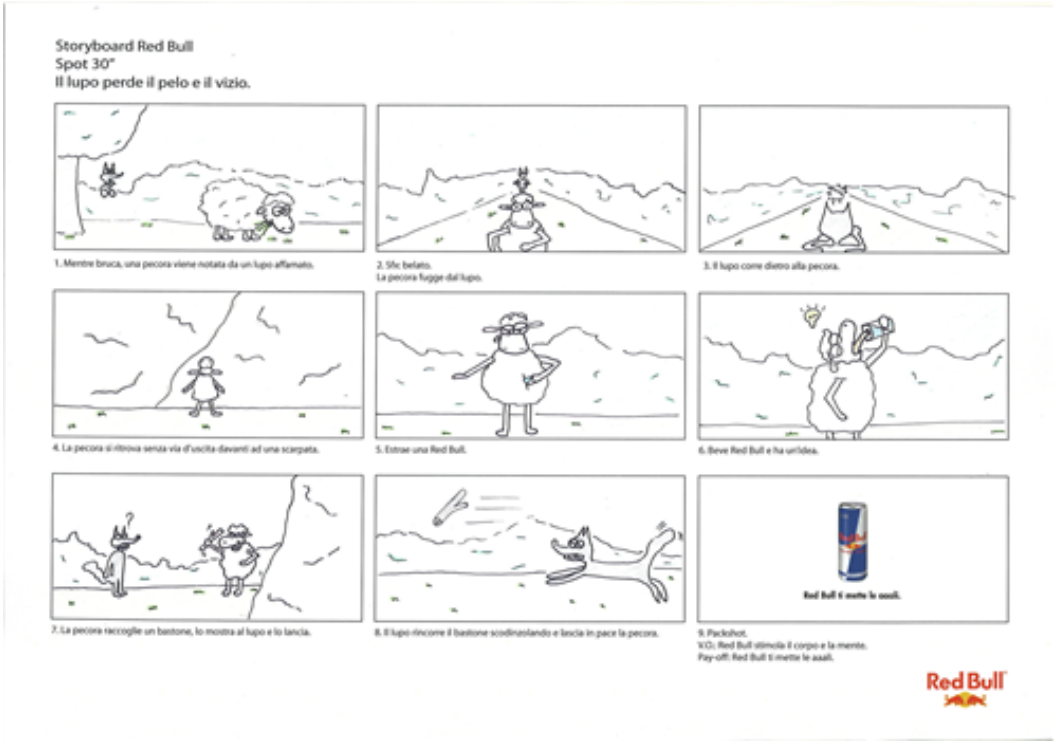


Figura 5.4: Storyboard realizzato per un contest della RedBull dal quale è stato preso spunto per questo caso d'uso [53]

In quest'ultimo è stato inserito un personaggio di tipo gruppo, così da mostrare come il sistema possa tollerarlo se, anche in questo caso, esso sia realizzato in modo consono. Un'altra differenza, che mostra una possibilità ulteriore di utilizzo, è che il piano che cade quando il cane ci passa sopra, salva delle informazioni per conto del cane ("il cane cade nelle foglie"), in quanto non è un oggetto del quale abbia senso scrivere nelle descrizione testuale della scena.

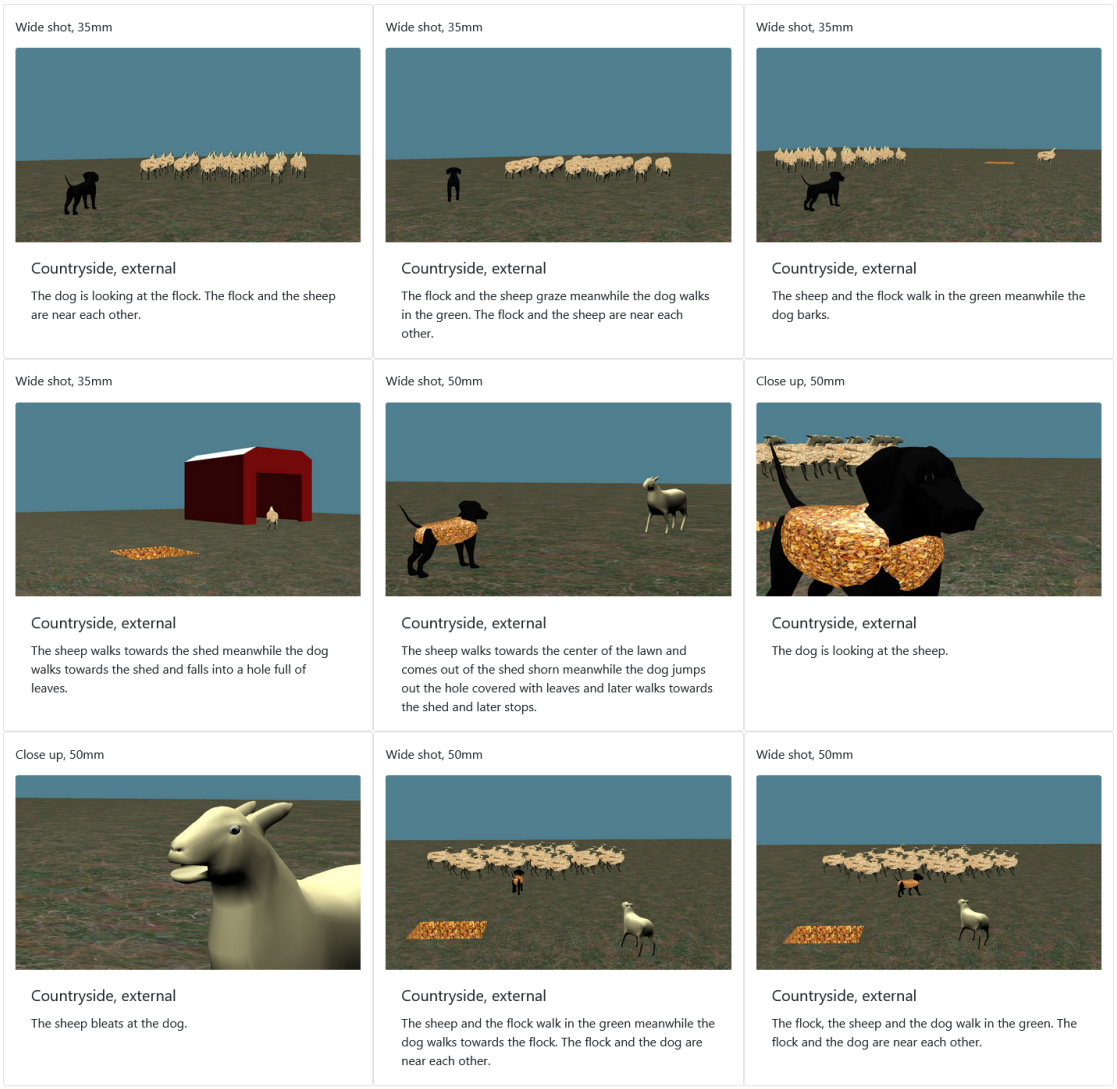


Figura 5.5: Storyboard del caso d'uso 3 realizzato con l'applicativo

5.2 Test

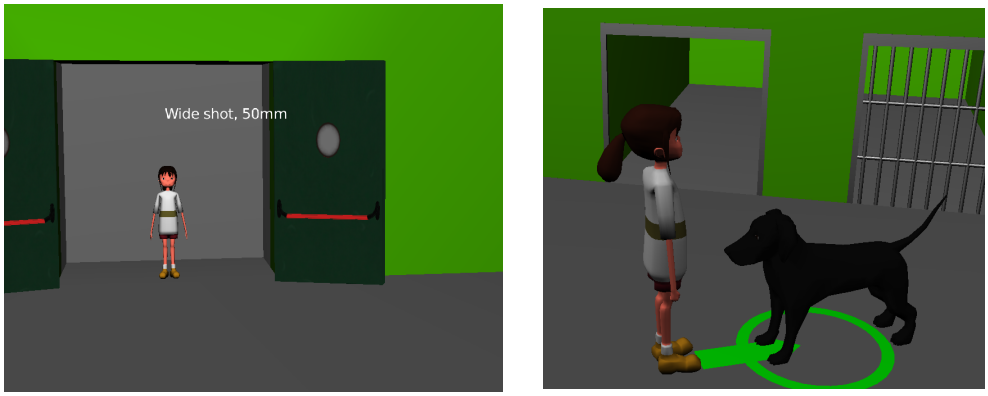
I test sono stati realizzati in due fasi: una prima ha permesso di identificare le maggiori lacune dal punto di vista di esperienza dell'utente, così da averle colmate per effettuare la seconda.

Queste le mancanze trovate dai primi test: non avere un feedback immediato nel momento in cui si salva un'inquadratura o si cambia la focale alla camera, non avere un indicazione visiva di quando si possono utilizzare i comandi che necessitano alcune condizioni per essere attivi o di quando il sistema rileva che si sta

camminando verso qualcosa, e il fatto che le camere (principale e dei personaggi) potessero attraversare i muri e che quella principale fosse in grado, potenzialmente, di muoversi all'infinito in una direzione, perdendo così di vista la scena creata.

La prima è stata risolta con l'aggiunta dell'oggetto testo come figlio della camera principale, il quale viene riempito con la descrizione della scena ogni qual volta si salvi o cancelli uno sketch. La stringa di testo che viene visualizzata è la stessa che si vede nello storyboard finale, così si ha per qualche istante l'inquadratura con il testo annesso di modo da fornire all'utente un immediato riscontro della descrizione creata (e poterla rifare se non soddisfacente). Tutto questo è reso possibile da *showText()*, appositamente realizzata. Allo stesso tempo, quando si cambia la focale della camera, l'oggetto testo mostra quale è inserita in quel momento e quale tipo di inquadratura si sta facendo (figura 5.6a).

La risoluzione della seconda è stata realizzata inserendo due frecce (una rossa e una verde) che alternandosi mostrano se si è diretti o meno verso un punto d'interesse, e con l'aggiunta di un cerchio alla base degli oggetti o personaggi che ne avevano bisogno, il quale diventa visibile quando il comando è attivo (figura 5.6b).



(a) (b)

Figura 5.6: Come sono state inserite le correzioni

Per quanto riguarda la terza, per la camera principale sono state aggiunte delle caratteristiche per rilevare le collisioni, di modo da bloccarla quando tocca un muro o un personaggio (per gli ambienti aperti, sono stati inseriti dei piani invisibili intorno al suolo). Nel dettaglio, vi è stata messa la fisica di tipo *Dynamic* con box collider *Cone*. Per le camere dei personaggi, come visto in precedenza, è stata invece aggiunta una seconda camera "in prima persona" (figura 5.7b) che diventa attiva quando l'altra, che ha una visione più ampia (figura 5.7a), viene ostruita da un oggetto, il quale viene rilevato grazie ad un ray sensor.

(a) (b)

Figura 5.7: Le due visuali del personaggio cane

Nella seconda fase di test, è stato prima sottoposto agli utenti un file di prova tramite il quale prendere dimestichezza con l'interfaccia, per poi passare al vero e proprio test, nel quale è stato utilizzato il primo caso d'uso con l'obiettivo di realizzare lo storyboard richiesto (vedi figura 5.1), fornendo all'utente come riferimento il seguente copione, appositamente scritto di modo che descrivesse l'intera scena così come era stata pensata.

```
1 INTERNAL, DOGHOUSE - DAY
2
3 The girl is at the entrance of the doghouse and is ready to find a
      dog to take home. The dog is in the cage.
4 She starts walking towards the cages and sees a black dog sitting
     in a cage.
5 She stops in front of that cage and the dog, realizing that
     something good can happen, stands up. They are now looking at
     each other.
6 The cage finally opens, so that the dog can reach the girl and she
      can caress him.
7 They walk towards the exit and near each other, and on their way
     to the exit, the dog barks happily.
8 They finally reach the exit and the dog is definitely out of the
     doghouse and has a new family. They are at the entrance, close
     to each other.
```

Alla fine del test, è stato sottoposto anche il questionario SUS (System Usability Scale) il quale permette di ottenere un valore in centesimi che rappresenta l'usabilità del sistema. Questo questionario è composto di 10 affermazioni alle quali si risponde dando un valore da 1 a 5. Esse hanno alternativamente un significato positivo (affermazioni dispari) o negativo (pari) e per calcolare il risultato finale si effettua quanto segue: per le frasi 1,3,5,7 e 9 il punteggio è pari al valore, da 1 a 5, espresso meno 1 (ad esempio, se si esprime il valore 4, quella affermazione avrà punteggio 3), mentre per le 2,4,6,8 e 10 il punteggio è 5 meno il valore espresso nella scala (se si esprime 3, si avrà punteggio 2). Ottenuti tutti i punteggi, questi

vengono sommati, e la loro somma moltiplicata per 2.5. Si ottiene un valore in centesimi che indica l'usabilità del sistema, la quale può essere considerata buona se questo valore è pari o superiore a 68 [54].

All'interno del questionario è stata inserita anche una sezione dove inserire commenti personali e lo storyboard realizzato, il quale in automatico, oltre alle card con i vari sketch, contiene anche una stringa con il tempo impiegato. Questo permette di fare un confronto tra gli utenti, sia dal punto di vista visivo che temporale (Appendice A.4).

Capitolo 6

Analisi risultati

Il test è stato effettuato da 12 studenti del corso di Computer Animation, presente nei corsi di Laurea Magistrale di Ingegneria Informatica, Ingegneria del Cinema e dei Mezzi di Comunicazione, e Design del Politecnico di Torino.

Tramite il questionario SUS, come visto in precedenza, è stato possibile valutare l'usabilità del sistema, il quale ha registrato una media di 79,17/100, piazzandosi dunque al di sopra della soglia di "buona usabilità". Si è registrato un picco negativo di valore 62,5 e due positivi con valore 87,5 (figura 6.1).

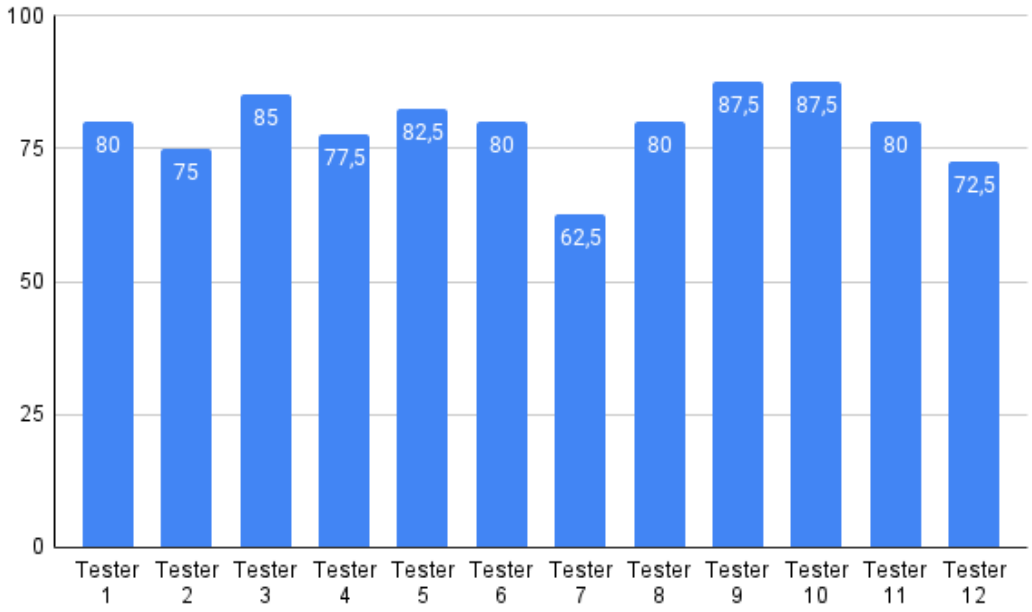


Figura 6.1: Grafico risultati SUS

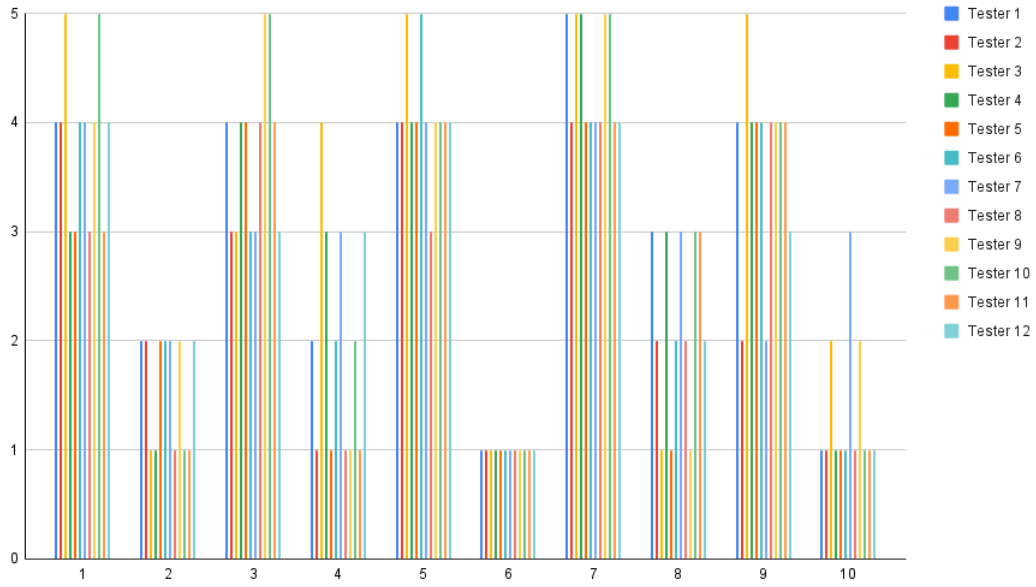


Figura 6.2: Andamento per ogni domanda

Oltre a questi dati è stato chiesto agli utenti, in maniera facoltativa, di inserire dei commenti su ciò che gli è piaciuto di più e di meno, nell'utilizzare il sistema. In generale, la parte positiva riguarda il fatto di riuscire ad ottenere uno storyboard rapidamente, facilmente e che sia rappresentativo di ciò che si voleva raccontare, oltre al fatto che il sistema rilevi bene le azioni effettuate dai personaggi e le interazioni fra loro. Dall'altro lato, ciò che ha convinto meno è stata la camera, la quale non risponde bene alle collisioni rimanendo spesso "incastrata" negli oggetti coi quali collide e anche per come è stato configurato il suo movimento nella scena (più tester hanno espresso la necessità di due tasti per abbassare/alzare la camera). Inoltre alcuni utenti hanno trovato i movimenti dei personaggi un po' macchinosi.

In linea generale, osservando i test e parlando con gli utenti a questionario consegnato, è emerso un generale apprezzamnto del lavoro svolto e del sistema creato, esprimendo alcuni, nel caso in cui se ne creasse un tool, la volontà di utilizzarlo.

Dal punto di vista temporale, gli storyboard sono stati realizzati con una media di 9 minuti e 24 secondi. In questi dati sono però da considerare due utenti che hanno deciso di rifare il test dall'inizio in quanto non soddisfatti del risultato ottenuto. Il secondo risultato, quindi, è realizzato con una maggiore confidenza dei comandi, la quale può aver portato ad una riduzione del tempo totale. Di conseguenza la media temporale, con solo il primo tentativo, risulterebbe maggiore. Il minor tempo impiegato è 4 minuti e 12 secondi (il quale è uno dei due casi in cui

lo storyboard è stato rifatto), mentre il maggiore è 19 minuti (figura 6.3). Questa grossa differenza di tempo impiegato è data da due fattori: una differente velocità del tester nel prendere confidenza con l'interfaccia e una maggiore o minore cura nello scegliere un'inquadratura soddisfacente.

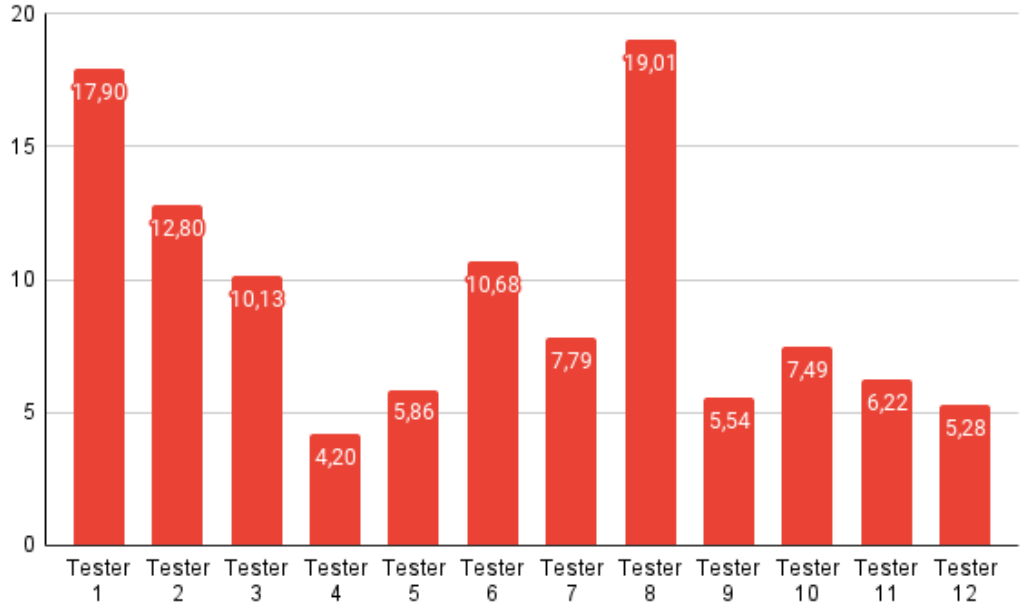


Figura 6.3: Tempo impiegato (minuti)

Andando invece più nel dettaglio, e analizzando il test dal quale è uscito l'unico valore al di sotto della soglia di usabilità, l'utente rientra nei due che hanno voluto rifare lo storyboard. Inoltre ha espresso, tra i pareri negativi, che avrebbe apprezzato l'avere un feedback di avvenuta cancellazione dell'ultima scena o un modo per vedere a run-time l'ultima salvata. Dall'altro lato, è rimasto soddisfatto del fatto che il sistema contemplasse le azioni contemporanee e anche dal punto di vista del tempo impiegato, rimane sotto la media con 7 minuti e 47 secondi. A margine di questo, ma non meno importante, anche il risultato finale è buono sia visivamente che per ciò che le descrizioni testuali create raccontano. Il punteggio finale si può quindi spiegare con il fatto che il risultato ottenuto la prima volta non è stato soddisfacente, e può aver portato ad una sensazione di minore usabilità generale.

Per quanto riguarda il risultato visivo, i 12 storyboard realizzati dai tester variano dai 5 ai 10 sketch realizzati (figura 6.4), mentre per quanto riguarda le inquadrature in sé, esse sono condizionate dal problema legato alla camera, per il

quale i tester a cui è capitato sono stati obbligati salvare degli shot dalla posizione in cui la camera era incastrata.

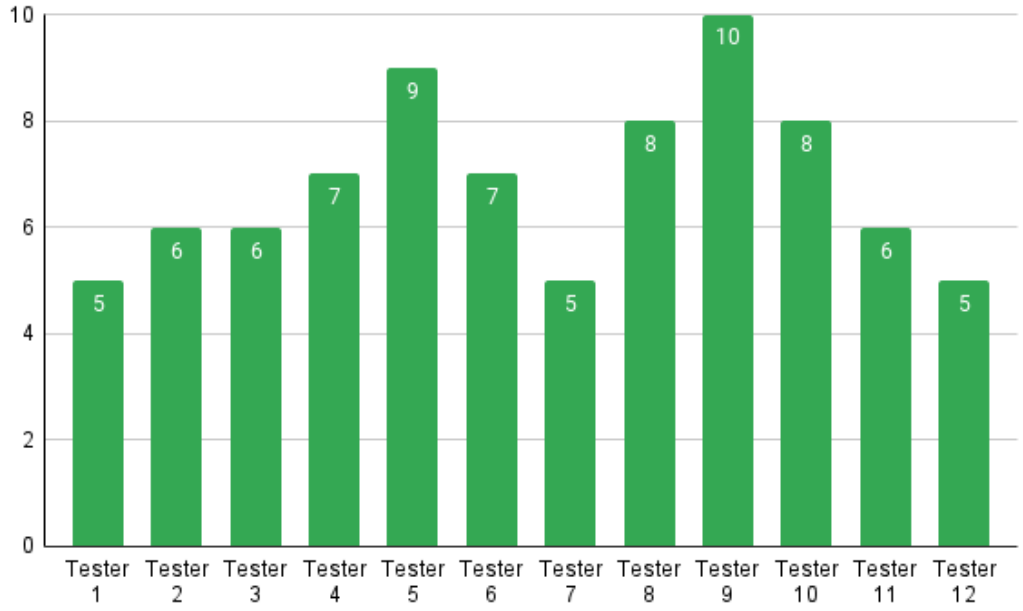


Figura 6.4: Sketch realizzati

Per le descrizioni testuali generate, invece, gli errori più comuni sono legati alle informazioni di direzione dei personaggi, e dalla poca confidenza con il sistema il quale, tenendo traccia di tutte le azioni effettuate, ha più volte generato frasi ridondanti con più azioni effettuate, le quali da copione non erano richieste. Più in generale, una poca attenzione ai vari dettagli inseriti nel copione ha portato ad alcune mancanze dal punto di vista del testo, così come forse una poca volontà di rifare uno sketch poco soddisfacente ha portato ad accettare il risultato anche se questo non era totalmente corretto.

Di seguito, i risultati ottenuti.

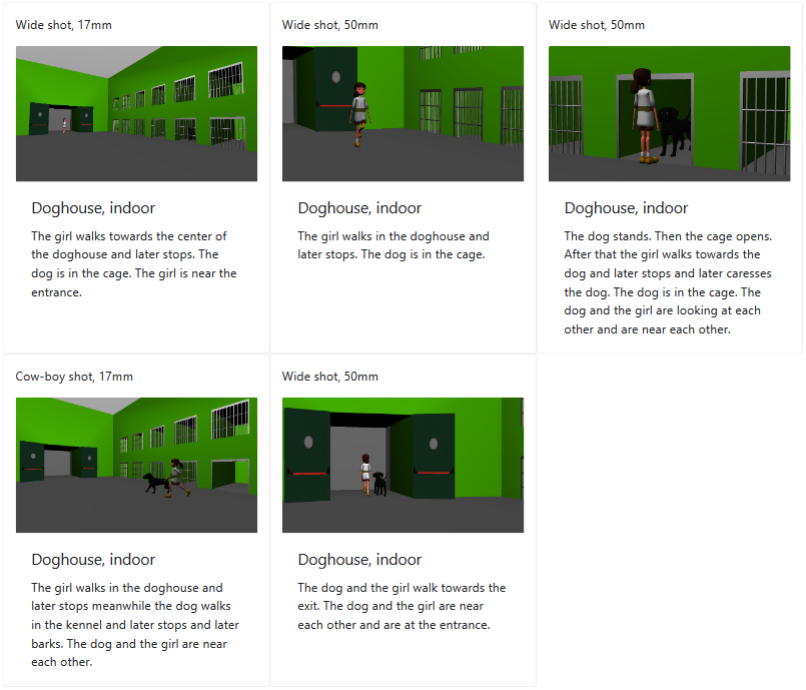


Figura 6.5: Storyboard tester 1

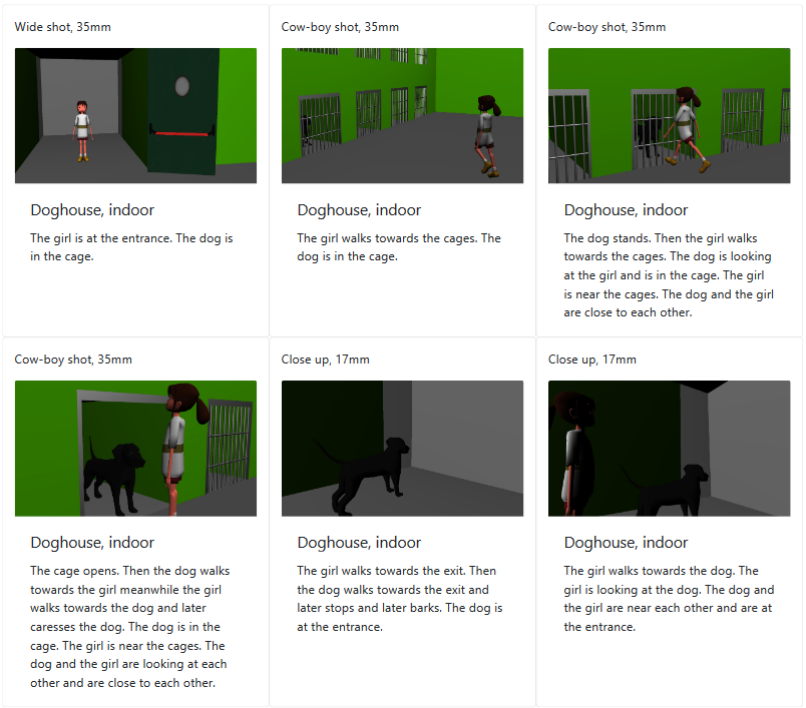


Figura 6.6: Storyboard tester 2

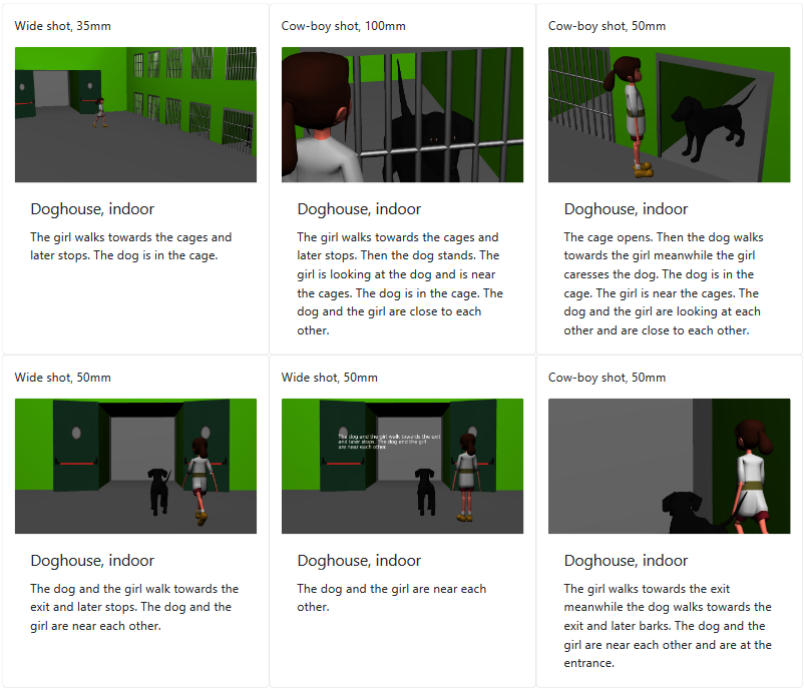


Figura 6.7: Storyboard tester 3

Figura 6.8: Storyboard tester 4

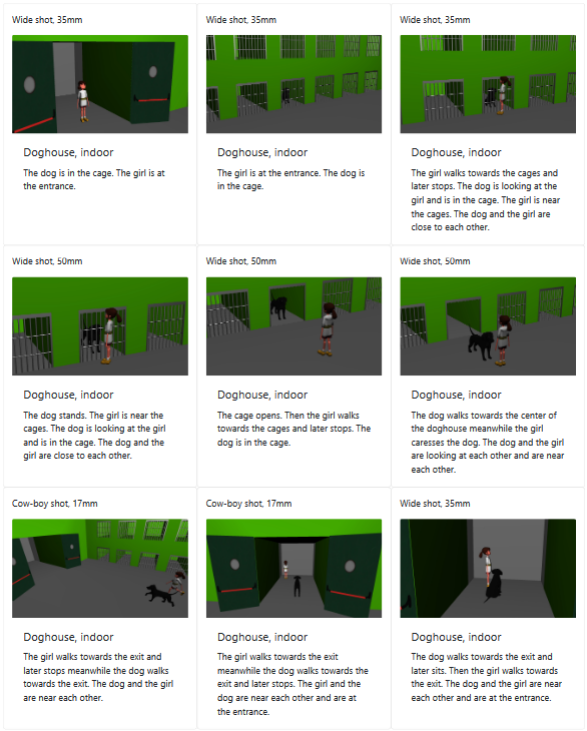


Figura 6.9: Storyboard tester 5

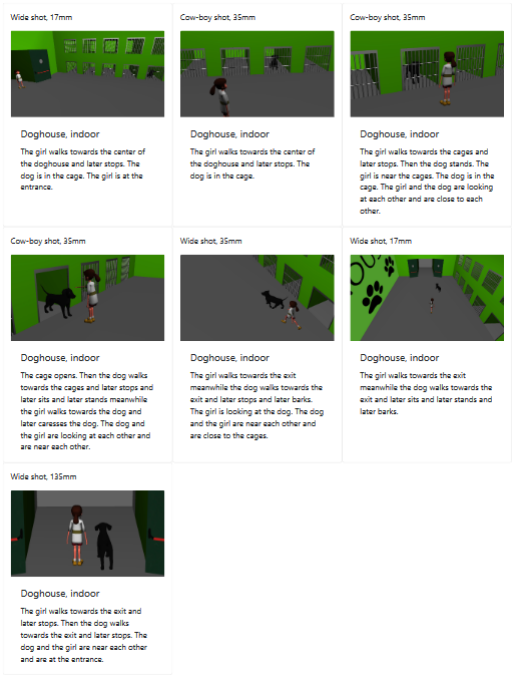


Figura 6.10: Storyboard tester 6

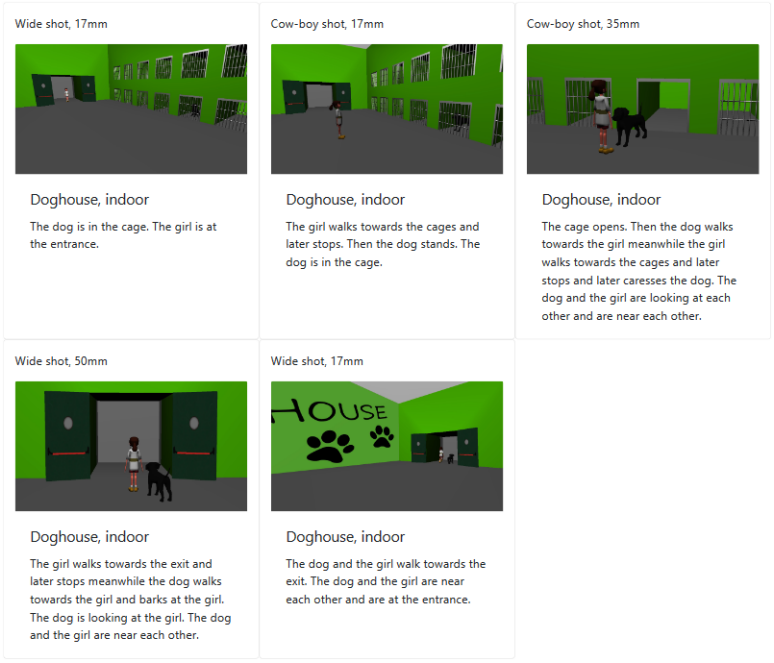


Figura 6.11: Storyboard tester 7

Figura 6.12: Storyboard tester 8

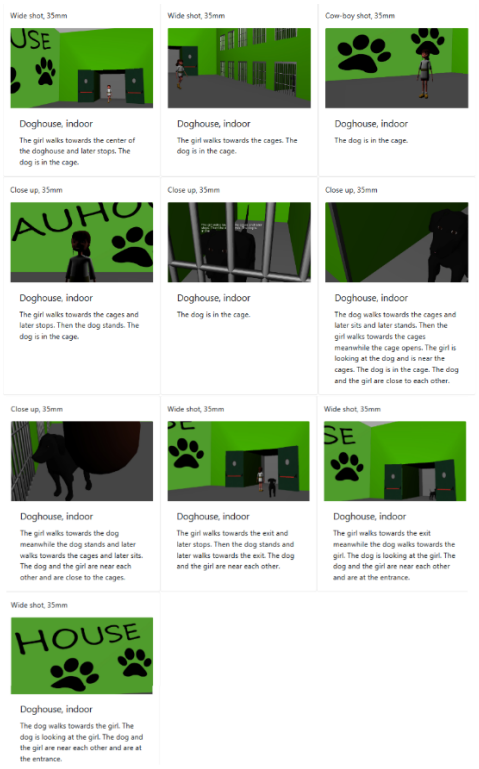


Figura 6.13: Storyboard tester 9

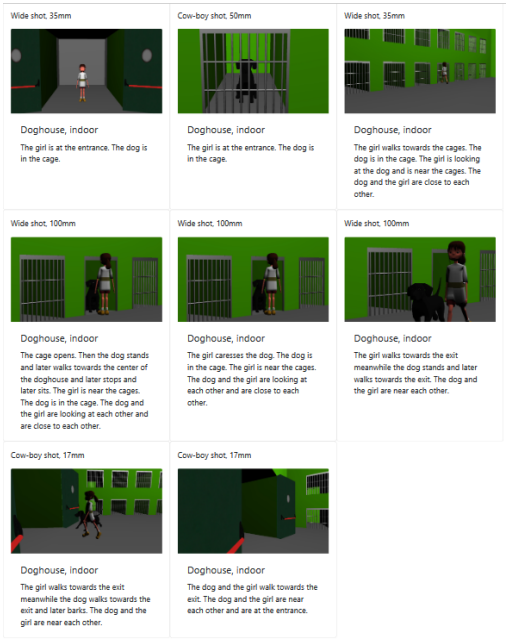


Figura 6.14: Storyboard tester 10

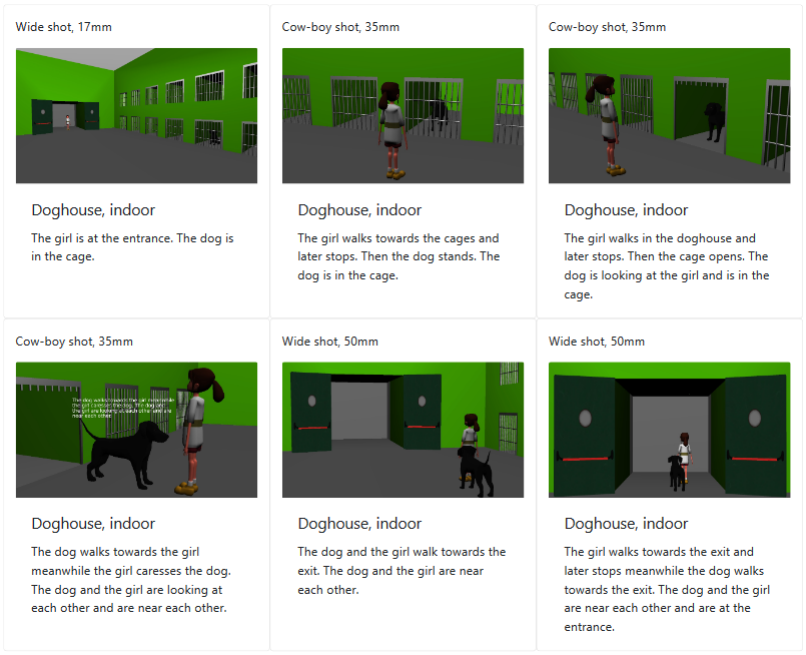


Figura 6.15: Storyboard tester 11

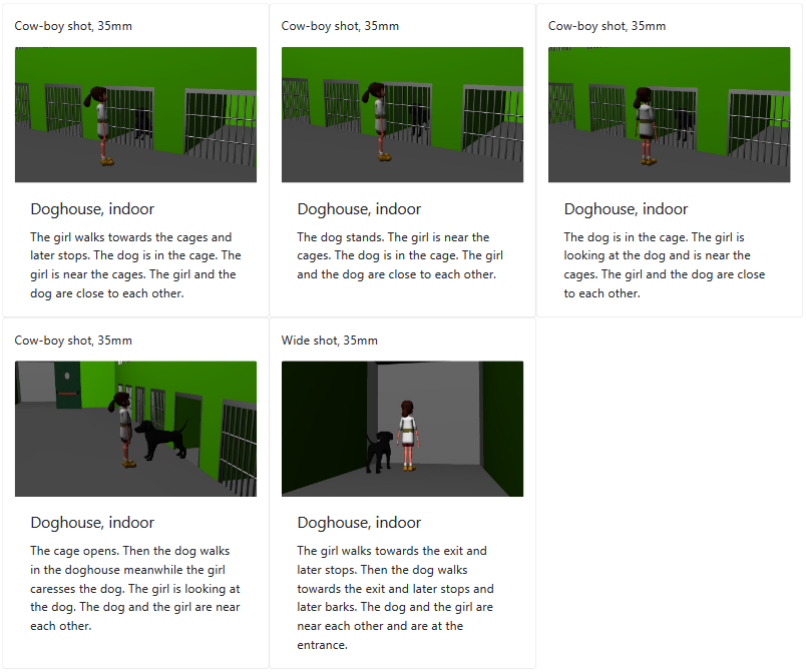


Figura 6.16: Storyboard tester 12

Capitolo 7

Conclusioni e sviluppi futuri

A partire dai test e dai risultati ottenuti, si evince come il sistema possieda un buon livello di usabilità e che esso sia, soprattutto per gli utenti appassionati o studiosi del settore, un buono strumento per poter realizzare in poco tempo uno storyboard che contempli un certo livello di dettaglio.

Le criticità emerse, invece, fanno da spunto per gli eventuali sviluppi futuri. Una su tutti, è la realizzazione di una camera più performante la quale, oltre a non portare più problemi dal punto di vista delle collisioni, fornisca anche altri parametri modificabili a run-time.

Lo sviluppo più naturale in conseguenza a questo lavoro, potrebbe essere quello di genereare un animatic invece dello storyboard. In questo caso, bisognerebbe specificare quando la camera deve registrare oppure no, in quanto spesso i movimenti (soprattutto di camminata) vengono corretti o rifatti per posizionare il personaggio in un punto migliore, e questo porterebbe l'eventuale animatic a delle tempistiche che non lo rappresenterebbero.

Un altro proseguimento potrebbe essere dare l'opportunità all'utente di comporre la scena in tempo reale aggiungendo oggetti presenti in una libreria precedentemente creata. Questo permetterebbe di creare ambientazioni ad hoc per ogni storia, ma avrebbe bisogno di una potenza di calcolo e intelligenza maggiore di quelle presenti nell'attuale applicativo.

Infine, rimanendo più vicini alla versione attuale, come emerso anche dai test, si potrebbero aggiungere eventuali feedback, come il poter vedere continuamente il risultato che si sta ottenendo o ciò che si può eliminare. Allo stesso tempo, il tipo e la quantità di informazioni interpretabili può essere ampliato notevolmente, inserendo altri dettagli come emozioni e dialoghi.

Tutto questo inoltre, potrebbe anche essere trasformato in un tool di Blender così che chiunque possa utilizzarlo in locale.

Appendice A

A.1 Creazione del file html

```
storyboard = open("Storyboard_scene1.html", "w")
storyboard.write(
'<head>'
    '<meta name="viewport" content="width=device-width, initial-scale=1">'
    '<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">'
    '<script src="https://unpkg.com/navigo@6"></script>'
'</head>'
    '<div class="container-fluid"> <div class="row"> '
)
storyboard.close()
```

A.2 Creazione e inserimento della card

```
storyboard = open("Storyboard_scene1.html", "a")
data_uri = base64.b64encode(open('C:\Screenshot'+str(own["counter"])+'_scene1.jpg', 'rb').read()).decode('utf-8')
img_tag = '<img class="card-img-top" src="{0}">'.format(data_uri)
card = (
'<div class="card col-sm-4" id="card'+ str(own["counter"]) +'" style="padding-top: 15px">' +
    '<p>' + camera["field"] + ", " + str(camera.lens) + "mm" + '</p>' +
    img_tag +
    '<div class="card-body">' +
        '<h5 class="card-title">' + "Doghouse, indoor" + '</h5>'
        '<p class="card-text">' + print_caption(own, [dog_arm, girl_arm, cage], [dog, girl]) + '</p>' +
    '</div>' +
```

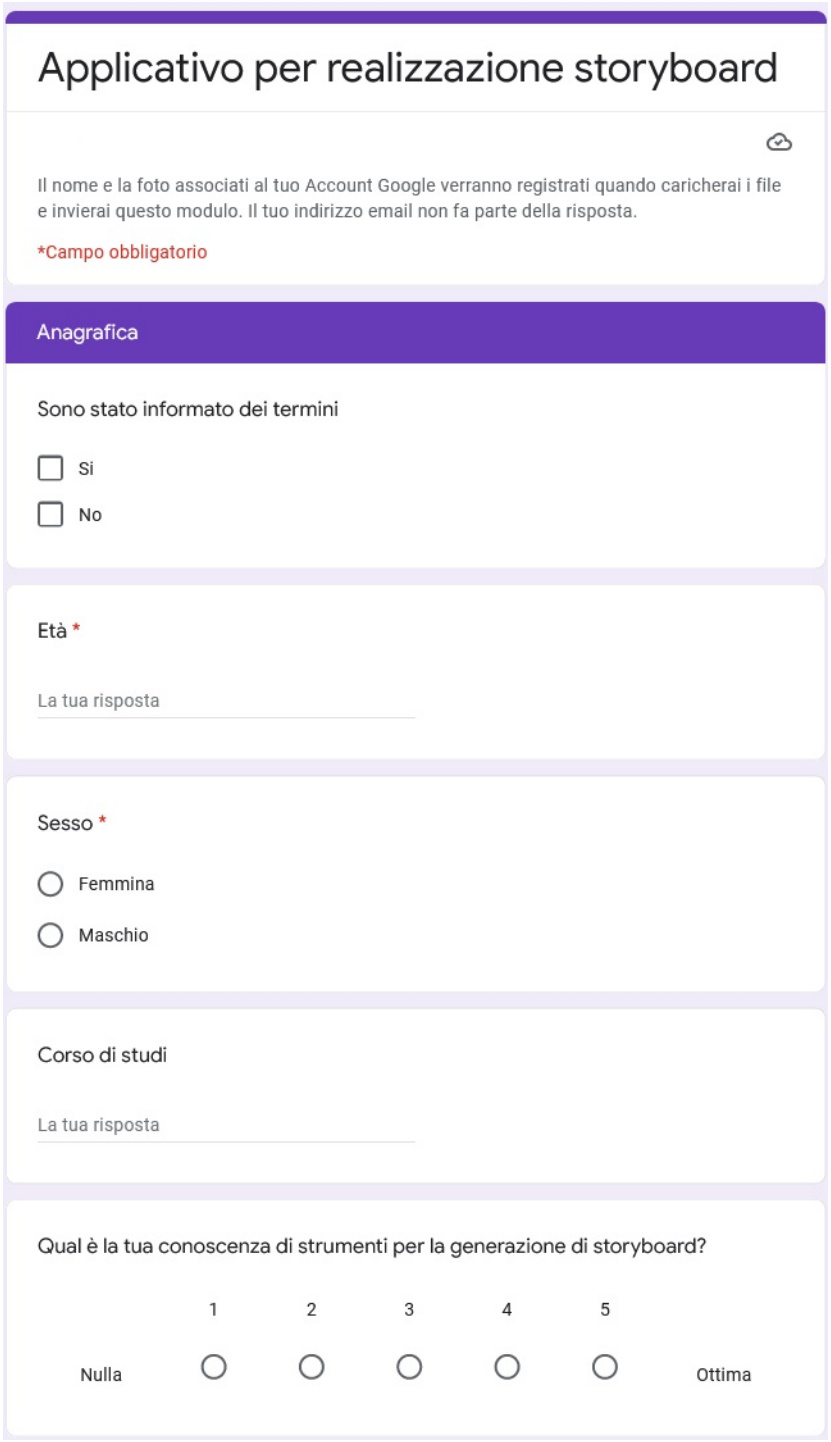
```
'</div>'
)
storyboard.write(card)
storyboard.close()
```

A.3 Cancellazione dell'ultima card salvata

```
storyboard = open("Storyboard_scene1.html", "r")
del_card = str(storyboard.read())
index = del_card.find('<div class="card col-sm-4" id="card'+ str(
    own["counter"]) +'"')
del_card = del_card[:index]
storyboard.close()

storyboard = open("Storyboard_scene1.html", "w")
storyboard.write(del_card)
storyboard.close()
```

A.4 Questionario



Applicativo per realizzazione storyboard

Il nome e la foto associati al tuo Account Google verranno registrati quando caricherai i file e invierai questo modulo. Il tuo indirizzo email non fa parte della risposta.

*Campo obbligatorio

Anagrafica

Sono stato informato dei termini

- [ ] Si
- [ ] No

Età *

La tua risposta

Sesso *

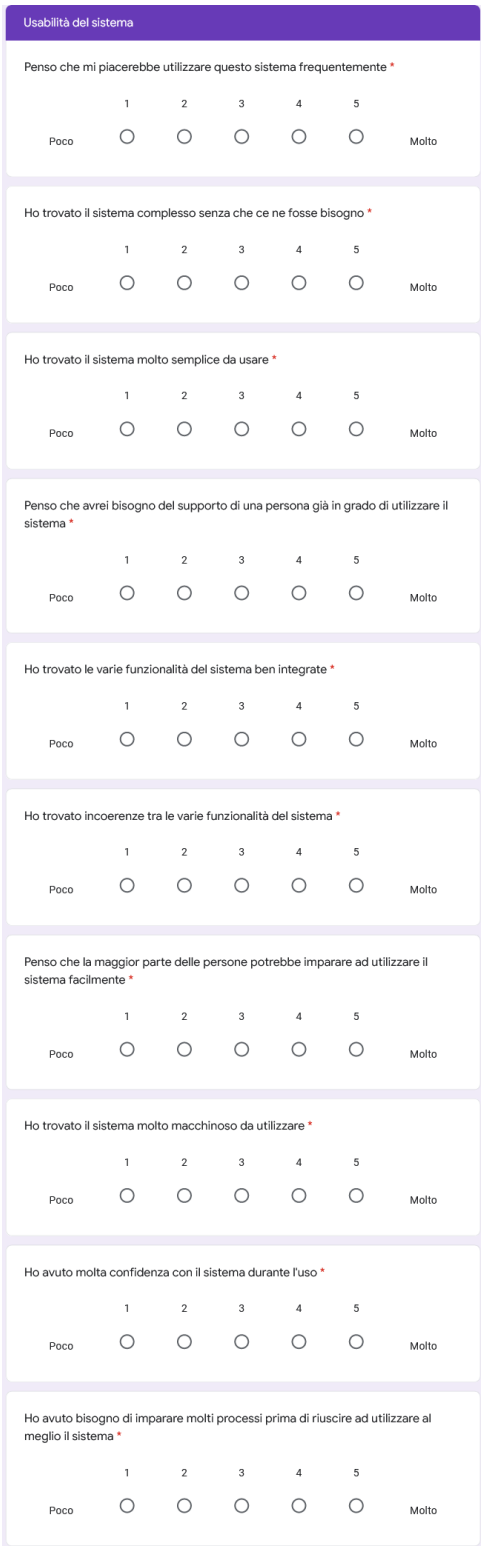
- ◯ Femmina
- ◯ Maschio

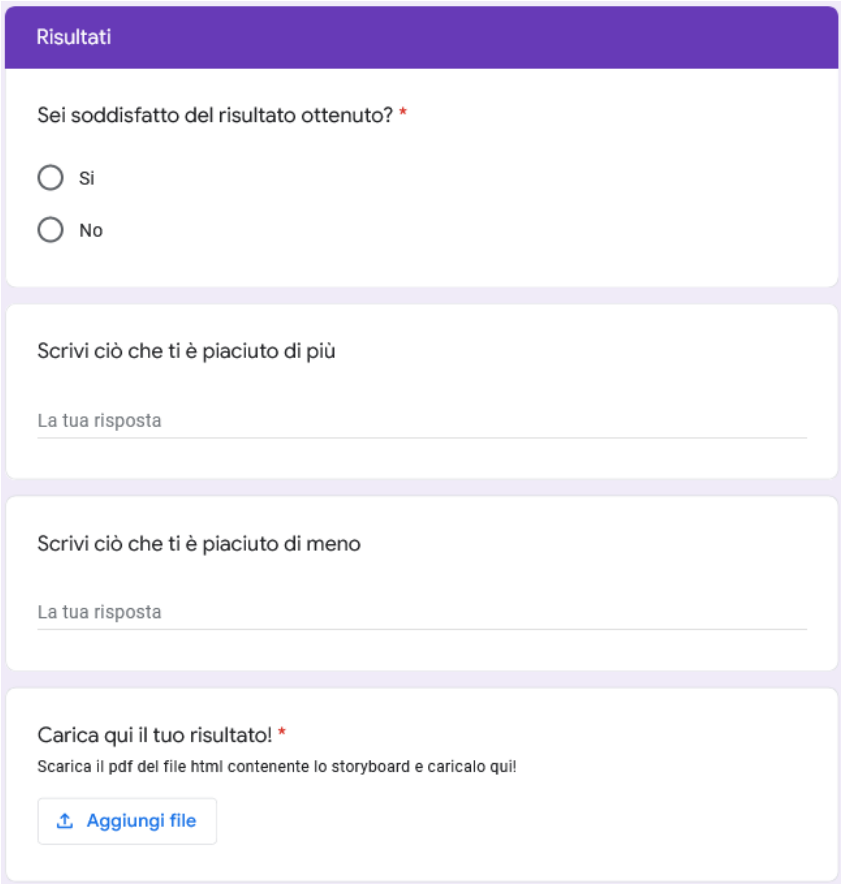
Corso di studi

La tua risposta

Qual è la tua conoscenza di strumenti per la generazione di storyboard?

	1	2	3	4	5	
Nulla	◯	◯	◯	◯	◯	Ottima


Usabilità del sistema
Penso che mi piacerebbe utilizzare questo sistema frequentemente *
1 2 3 4 5
Poco
Molto
Ho trovato il sistema complesso senza che ce ne fosse bisogno *
1 2 3 4 5
Poco
Molto
Ho trovato il sistema molto semplice da usare *
1 2 3 4 5
Poco
Molto
Penso che avrei bisogno del supporto di una persona già in grado di utilizzare il sistema *
1 2 3 4 5
Poco
Molto
Ho trovato le varie funzionalità del sistema ben integrate *
1 2 3 4 5
Poco
Molto
Ho trovato incoerenze tra le varie funzionalità del sistema *
1 2 3 4 5
Poco
Molto
Penso che la maggior parte delle persone potrebbe imparare ad utilizzare il sistema facilmente *
1 2 3 4 5
Poco
Molto
Ho trovato il sistema molto macchinoso da utilizzare *
1 2 3 4 5
Poco
Molto
Ho avuto molta confidenza con il sistema durante l'uso *
1 2 3 4 5
Poco
Molto
Ho avuto bisogno di imparare molti processi prima di riuscire ad utilizzare al meglio il sistema *
1 2 3 4 5
Poco
Molto


Risultati
Sei soddisfatto del risultato ottenuto? *
Si
No
Scrivi ciò che ti è piaciuto di più
La tua risposta
Scrivi ciò che ti è piaciuto di meno
La tua risposta
Carica qui il tuo risultato! *
Scarica il pdf del file html contenente lo storyboard e caricalo qui!
Aggiungi file

Bibliografia

[1] Wikipedia. *Storyboard*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/actuators/types/property.html.

[2] Deviantart. *LionKing/storyboards*. URL: https://www.deviantart.com/amberhollinger/art/LionKing-storyboards-337333235.

[3] Ryzom CC BY-SA 2.0. In: *Wikimedia Commons* (2014). URL: https://creativecommons.org/licenses/by-sa/2.0.

[4] Khai N. Truong, Gillian R. Hayes e Gregory D. Abowd. "Storyboarding: an empirical determination of best practices and effective guidelines". In: *Proceedings of the 6th Conference on Designing Interactive Systems*. 2006, pp. 12–21.

[5] Niina Kantola e Timo Jokela. "SVSb: simple and visual storyboards: developing a visualisation method for depicting user scenarios". In: *Proceedings of the 19th Australasian Conference on Computer-Human Interaction: Entertaining User Interfaces*. 2007, pp. 49–56.

[6] Mieke Haesen et al. "Draw Me a Storyboard: Incorporating Principles & Techniques of Comics..." In: *Proceedings of HCI 2010 24* (2010), pp. 133–142.

[7] Angelo EM Ciarlini et al. "Event relations in plan-based plot composition". In: *Computers in Entertainment (CIE)* 7.4 (2010), pp. 1–37.

[8] *TVPaint website*. URL: https://www.tvpaint.com/.

[9] *ToonBoom Storyboard website*. URL: https://www.toonboom.com/products/storyboard-pro.

[10] *Storyboard That website*. URL: https://www.storyboardthat.com/it.

[11] *Cine Tracer website*. URL: https://www.cinetracer.com/.

[12] P. P. Mohanta, Saha S. K. e B. Chanda. "Generation of size constrained video storyboard using spanning tree". In: *Proceedings of the First International Conference on Internet Multimedia Computing and Service*. 2009, pp. 179–182.

[13] Shizhe Chen et al. "Neural Storyboard Artist: Visualizing Stories with Coherent Image Sequences". In: *Proceedings of the 27th ACM International Conference on Multimedia*. 2019, pp. 2236–2244.

[14] Robert Held et al. "3D puppetry: a kinect-based interface for 3D animation". In: *UIST*. Vol. 12. Citeseer. 2012, pp. 423–434.

[15] Hariharan Subramonyam et al. "TakeToons: Script-driven Performance Animation". In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 2018, pp. 663–674.

[16] Yeyao Zhang et al. "Generating Animations from Screenplays". In: *arXiv preprint arXiv:1904.05440* (2019).

[17] Blender Italia. *Conoscenza*. URL: `https://www.blender.it/conoscenza/`.

[18] Williams Richard. *The Animator's Survival Kit*. New York: Farrar, Straus e Giroux, 2009.

[19] Wikipedia. *Blender Game Engine*. URL: `https://it.wikipedia.org/wiki/Blender_Game_Engine`.

[20] Blender 2.79 Reference Manual. *Sensors Introduction*. URL: `https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/introduction.html`.

[21] Blender 2.79 Reference Manual. *Controllers Introduction*. URL: `https://docs.blender.org/manual/en/2.79/game_engine/logic/controllers/introduction.html`.

[22] Blender 2.79 Reference Manual. *Actuators Introduction*. URL: `https://docs.blender.org/manual/en/2.79/game_engine/logic/actuators/introduction.html`.

[23] Blender 2.79 Reference Manual. *Sensors Types*. URL: `https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/index.html#sensor-types`.

[24] Blender 2.79 Reference Manual. *Controllers Types*. URL: `https://docs.blender.org/manual/en/2.79/game_engine/logic/controllers/index.html#controller-types`.

[25] Blender 2.79 Reference Manual. *Actuators Types*. URL: `https://docs.blender.org/manual/en/2.79/game_engine/logic/actuators/index.html#actuator-types`.

[26] Blender 2.79 Reference Manual. *Always Sensor*. URL: `https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/always.html`.

[27] Blender 2.79 Reference Manual. *Collision Sensor*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/collision.html.

[28] Blender 2.79 Reference Manual. *Delay Sensor*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/delay.html.

[29] Blender 2.79 Reference Manual. *Keyboard Sensor*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/keyboard.html.

[30] Blender 2.79 Reference Manual. *Mouse Sensor*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/mouse.html.

[31] Blender 2.79 Reference Manual. *Near Sensor*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/near.html.

[32] Blender 2.79 Reference Manual. *Property Sensor*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/property.html.

[33] Blender 2.79 Reference Manual. *Radar Sensor*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/radar.html.

[34] Blender 2.79 Reference Manual. *Ray Sensor*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/sensors/types/ray.html.

[35] Blender 2.79 Reference Manual. *AND Controller*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/controllers/types/and.html.

[36] Blender 2.79 Reference Manual. *OR Controller*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/controllers/types/or.html.

[37] Blender 2.79 Reference Manual. *Python Controller*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/controllers/types/python.html.

[38] Blender 2.79 Reference Manual. *Edit Object Actuator*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/actuators/types/edit_object.html.

[39] Blender 2.79 Reference Manual. *Mouse Actuator*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/actuators/types/mouse.html.

[40] Blender 2.79 Reference Manual. *Motion Actuator*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/actuators/types/motion.html.

[41] Blender 2.79 Reference Manual. *Property Actuator*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/actuators/types/property.html.

[42] Blender 2.79 Reference Manual. *Visibility Actuator*. URL: https://docs.blender.org/manual/en/2.79/game_engine/logic/actuators/types/visibility.html.

[43] Python Italia. *Cos'è Python*. URL: https://www.python.it/about/.

[44] Blender 2.79 API Documentation. *Python in Blender*. URL: https://docs.blender.org/api/2.79/info_overview.html.

[45] Blender 2.79 API Documentation. *Game Logic (bge.logic)*. URL: https://docs.blender.org/api/2.79/bge.logic.html.

[46] Python.org. *math*. URL: https://docs.python.org/3/library/math.html.

[47] Python.org. *re*. URL: https://docs.python.org/3/library/re.html.

[48] Python.org. *time*. URL: https://docs.python.org/3/library/time.html.

[49] Python.org. *base64*. URL: https://docs.python.org/3/library/base64.html.

[50] Python.org. *webbrowser*. URL: https://docs.python.org/3/library/webbrowser.html.

[51] Python.org. *os*. URL: https://docs.python.org/3/library/os.html.

[52] Pinterest. *LILO and STITCH 2 STORYBOARDS*. URL: https://www.pinterest.it/pin/63050463514473348/.

[53] IED. *Scelti i vincitori del Red Bull Spot Contest!* URL: https://www.ied.it/blog/scelti-i-vincitori-del-red-bull-spot-contest/1239.

[54] John Brooke. "SUS: A quick and dirty usability scale". In: *Usability Eval. Ind.* 189 (nov. 1995).