POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Tesi di Laurea Magistrale in Ingegneria Informatica

Produzione di contenuti grafici interattivi mediante strumenti di editing visuale

Relatori:
Prof. Fabrizio Lamberti
Prof. Andrea Sanna

Candidato:
Federico Salaroglio

Sessione di Ottobre 2017

Indice

Indice delle figure

Capitolo 1

Introduzione

In questo capitolo si introduce il lavoro di tesi svolto, definendo il contesto nel quale si colloca e gli obiettivi che ci si è prefissati di raggiungere. In seguito viene fornita una descrizione della struttura dell'elaborato, illustrando la metodologia con la quale ci si è approcciati nello sviluppare il lavoro.

1.1 Contesto

La tecnologia si diffonde sempre di più nella vita delle persone, in ogni ambito, influenzando qualunque attività quotidiana. Le applicazioni informatiche diventano utili strumenti per proporre un proprio prodotto ai clienti, pubblicizzare un esercizio commerciale o intrattenere le persone con attività ludiche o educative. Negli ultimi anni si assiste a un impiego crescente di nuove tecnologie per l'interazione con queste applicazioni. L'obiettivo è quello di riuscire ad immergere in misura sempre maggiore gli utenti, coinvolgendoli e cercando di rendere più interessante l'esperienza loro proposta. Un valido esempio è rappresentato dal contributo all'apprendimento dato dall'utilizzo di nuove tecnologie di comunicazione nelle scuole.

L'interazione uomo-macchina (HCI, Human-Computer Interaction) [1] tratta appunto la progettazione e utilizzo di tecnologie focalizzate sull'interazione tra le persone e i computer. Osservando il modo in cui gli umani interagiscono con i computer, si cerca di sviluppare nuovi metodi per interfacciarsi ad essi, attraverso lo sviluppo di sistemi interattivi che risultino affidabili, di facile utilizzo e che siano di supporto per le attività umane [2]. Il termine interazione uomo-macchina divenne popolare a partire dal 1983, grazie alla pubblicazione del libro "*The Psychology of Human-Computer Interaction*", di Stuart K. Card, Allen Newell e Thomas P. Moran [3], dove venivano illustrate le prime teorie e i risultati in questo settore. Successivamente, nel 1985 Kieras e Polson contribuirono all'analisi cognitiva delle interfacce uomo-macchina pubblicando "*The Acquisition and Performance of Text-Editing Skill: A Cognitive Complexity Analysis*" [4].

L'interazione nel tempo ha avuto una costante evoluzione: se agli albori vi era la sola interfaccia a riga di comando (CLI), il salto di qualità si ebbe grazie all'interfaccia grafica utente (GUI) e si ritiene che la prossima evoluzione sia dare dalle cosiddette interfacce naturali (NUI, Natural User Interface), ovvero

quelle totalmente invisibili, prive di dispositivi di controllo da dover apprendere e perciò estremamente intuitive.

Oltre alle tecnologie impiegate, si devono considerare i diversi contesti d'uso possibili e gli obiettivi degli utenti. In particolare, in ambienti museali o espositivi in genere si studiano sempre nuovi metodi per migliorare l'esposizione dei contenuti proposti, al fine di catturare l'interesse del visitatore. Il contenuto presentato diventa non più un mero oggetto statico in vetrina, ma assume sempre maggiormente una caratteristica dinamica, in grado di reagire in maniera opportuna a come gli utenti si approcciano a esso; a loro volta, gli utenti assumono un ruolo partecipe e attivo rispetto ai contenuti.

Esempi di queste nuove forme di interazione possono andare dai più comuni dispositivi tattili (ad esempio lo schermo di un moderno smartphone) per arrivare a soluzioni basate sul riconoscimento di movimenti del corpo, del viso o di comandi vocali, fino ad esperienze di realtà aumenta o virtuale (anche se queste ultime possono risultare meno idonee a un ambiente aperto al pubblico come può essere un museo, in quanto richiedono al singolo fruitore di indossare visori o altre attrezzature tendendo probabilmente ad isolarlo anziché favorire una socializzazione con gli altri visitatori). Inoltre, esistono i cosiddetti sistemi multimodali, che sono strumenti informatici in grado di supportare un'interazione attraverso più canali di comunicazione, ma si distinguono da quelli multimediali per la capacità di ricavare un'informazione unica e coerente tra i vari canali.

In un tale contesto, i curatori di un museo o un'esposizione ed i loro collaboratori devono adattarsi a queste evoluzioni. Spesso è richiesto loro di possedere competenze informatiche atte a permettere loro di avere il pieno controllo di queste nuove tecnologie per poter contribuire alla realizzazione di esposizioni sempre più efficaci. Ciò può limitare la capacità di intervenire autonomamente sugli strumenti utilizzati. Pertanto, è importante disporre di applicazioni semplici da utilizzare, che diano in linea di principio la possibilità, alle diverse categorie di utenti interessate, di realizzare in modo intuitivo e soddisfacente le esperienze che intendono proporre. Tali applicazioni devono potersi adattare ai diversi contesti di utilizzo, tenendo in considerazione gli obiettivi ma anche le attitudini, le abilità e le competenze degli utenti. Strumenti rivolti, ad esempio, all'apprendimento e per un pubblico di giovanissimi dovrebbero risultare più accattivanti e puliti nella grafica, mentre quelli rivolti ad un utilizzo lavorativo potrebbero necessitare di una maggiore gamma di funzioni.

1.2 Obiettivi

L'obiettivo del presente lavoro di tesi è quello di realizzare un programma di editing per la rappresentazione di contenuti tridimensionali interattivi in scenari espositivi, puntando a migliorare le

prestazioni di uno strumento software (applicazione) denominato “Leap Embedder” e sviluppato in un precedente lavoro [5]. Lo strumento di riferimento, realizzato anch’esso nell’ambito di una tesi di laurea, è stato pensato per permettere a utenti senza alcuna particolare competenza di realizzare simulazioni interattive (scene) a partire da una serie di oggetti grafici realizzati da esperti in un sistema di modellazione/animazione.

Le scene ottenute sono (erano) pensate per essere visualizzate in una teca olografica (3D) in un’esposizione museale, ma possono essere fruite anche attraverso un normale schermo (2D). Per interagire con i contenuti, gli utenti possono utilizzare approcci diversi, dai tradizionali mouse e tastiera fino ad interfacce naturali basate, ad esempio, sul riconoscimento di pose e gesti delle mani. Nel caso specifico, la teca olografica utilizzata nel lavoro citato è dotata di un sistema di interazione gestuale realizzato utilizzando il Leap Motion Controller, un dispositivo per il tracciamento delle mani. Come detto, le simulazioni visualizzate in teche come quella in esame sono pensate principalmente per ambiti museali, pubblicitari o comunque laddove sia richiesta una componente espositiva in cui un utente possa avere un’interazione naturale con i contenuti.

Nel presente lavoro si è deciso di sfruttare le interfacce già disponibili con la teca olografica ed il relativo dispositivo di tracciamento per dedicarsi alla realizzazione di un nuovo strumento software in grado di dialogare con un tool per la modellazione/l’animazione open-source (nello specifico, Blender) al fine di ottenere le informazioni relative agli oggetti tridimensionali da utilizzare nelle scene, di permettere l’utilizzo di tali oggetti per la realizzazione della simulazione tramite la definizione di un’opportuna logica di interazione e di consentire infine di esportare quanto creato per la visualizzazione (nello specifico, per la visualizzazione si sfrutta nuovamente Blender, in particolare il suo motore grafico real-time noto come Blender Game Engine, BGE).

L’applicazione realizzata mira a semplificare i passi necessari per la creazione della logica di interazione rispetto allo strumento di riferimento, ma anche rispetto a Blender stesso. Infatti, Blender mette a disposizione un proprio editor di scene, noto come Logic Editor, che presuppone, da un lato, una buona familiarità dell’utilizzatore con le funzionalità di modellazione/animazione integrate e, dall’altro, una discreta conoscenza dei costrutti di programmazione, rendendo inaccessibile tale soluzione alle categorie di utenti considerate in questo lavoro.

La nuova applicazione, denominata “Visual Scene Editor”, si pone come obiettivo quello di risultare più intuitiva rispetto a quelle esistenti. Si ritiene, infatti, che lo strumento precedentemente realizzato non si allontanasse sufficientemente da alcuni princìpi alla base dell’editor integrato di Blender, ereditandone quindi, almeno in parte, la complessità. Ad esempio, il focus dello strumento di riferimento era fortemente incentrato sul concetto di oggetto, rendendo difficile all’utente comprendere le relazioni con altri elementi della scena. Il nuovo strumento mira quindi ad astrarre quanto più possibile il funzionamento del Logic Editor, puntando ad ottenere un’interfaccia efficiente e immediata con la quale un qualsiasi utente, senza conoscenze specifiche di Blender o della programmazione, possa interagire

senza particolari difficoltà nell'apprendimento iniziale. Anche un utente esperto nell'utilizzo di strumenti grafici dovrebbe poter trovare vantaggi dall'uso della soluzione proposta, ad esempio in termini di un risparmio del tempo necessario per realizzare una determinata scena. Ci si aspetta, al contempo, che lo strumento realizzato sia sufficientemente flessibile da consentire future espansioni ed adattamenti a differenti ambiti applicativi e ad altre tipologie di input.

1.3 Organizzazione dei contenuti

Il presente documento si articola in sei capitoli. Terminata l'iniziale introduzione sul contesto e gli obiettivi che ci si era prefissati, si procede a riportare, nel secondo capitolo, un'ampia panoramica dello stato dell'arte nell'ambito dei linguaggi di programmazione visuali, focalizzando l'attenzione su tre di essi che più si avvicinano al risultato desiderato, per poi descrivere nel terzo capitolo le tecnologie hardware e software che sono state impiegate. Il quarto capitolo riguarda lo specifico lavoro svolto nella tesi; saranno quindi trattati nel dettaglio la progettazione e il funzionamento dell'applicazione creata per la definizione della logica di interazione e l'add-on realizzato per l'esportazione di tale logica al sistema di visualizzazione. Il quinto capitolo proporrà un caso di studio utile per verificare il funzionamento dell'applicazione ed illustrerà i risultati ottenuti da una serie di test di usabilità condotti con utenti. Per ultimo, il sesto capitolo esporrà le conclusioni ed analizzerà quali potrebbero essere gli eventuali sviluppi futuri.

Capitolo 2

Stato dell'arte

Per progettare l'applicazione Visual Scene Editor, si è partiti eseguendo un'analisi degli strumenti già esistenti in materia. Dato che la nuova applicazione si prefigge lo scopo di semplificare l'utilizzo del Blender Game Engine, così da renderlo fruibile in modo chiaro e intuitivo ad ogni utente, si può asserire che si desidera ottenere un prodotto "for dummies", perciò si è preso spunto soprattutto dai linguaggi di programmazione utilizzati in ambiti educativi (ad esempio con i bambini). Questi sono progettati principalmente come strumento di apprendimento, anziché per un uso professionale, e tendono ad astrarre il codice secondo una logica di "condizione-azione", in maniera molto simile a ciò che si trova nel BGE relativamente al concetto di sensori e attuatori. Tra i linguaggi di programmazione educativi vi sono dei veri e propri strumenti pensati per l'insegnamento della programmazione agli studenti o ai bambini ed anche dei videogiochi per PC e console. Negli anni sono stati svolti alcuni esperimenti con strumenti che permettessero al giocatore di definire il proprio personale videogioco, gestendone le logiche di interazione attraverso un'interfaccia utente (Graphics User Interface, GUI) che si rivelasse quanto più intuitiva possibile, ma al contempo sufficientemente versatile e profonda da concedere la massima libertà di personalizzazione.
In questo capitolo si inizierà con un'ampia panoramica sui linguaggi di programmazione visuali per poi concentrarsi nel dettaglio sui tre programmi che più si avvicinano alle necessità del lavoro: Scratch, Kodu e Project Spark.

2.1 Linguaggi di programmazione visuale

Fin dalle origini gli umani hanno prediletto utilizzare le immagini per comunicare tra loro, a partire dalle pitture rupestri degli uomini preistorici, passando per i geroglifici egizi. Inoltre le persone tendono a pensare e ricordare i concetti in termini di immagini. Pertanto è prevedibile che un linguaggio di programmazione non testuale possa essere accessibile a un maggior numero di persone rispetto a uno testuale, risultando meno complicato da apprendere e da utilizzare efficacemente. Ridurre o rimuovere completamente la necessità di trasferire le idee visuali in una qualche rappresentazione testuale artificiale può aiutare a mitigare il problema di questa ripida curva di apprendimento. D'altronde quando

un programmatore deve provare a spiegare il funzionamento di un suo programma, il metodo più utilizzato è quello di ricorrere ad una rappresentazione grafica su carta o su lavagna, dove attraverso un flusso di controllo, con dei box e delle frecce, si cerca di mostrare il comportamento del programma in modo semplice e comprensibile a chiunque. Data la convenienza di questo tipo di rappresentazione, fin dagli anni '70 si è pensato di permettere di poter scrivere direttamente i programmi in questo modo, anche perché molte applicazioni ben si prestano a metodi di sviluppo visuali.

Un linguaggio di programmazione visuale (Visual Programming Language, VPL) è un linguaggio che consente la programmazione per mezzo di una manipolazione grafica degli elementi, anziché tramite sintassi scritta [6]. Un VPL permette di programmare con "espressioni visuali" (arrangiamenti spaziali di testo e simboli grafici) ed eventualmente anche di inserire spezzoni di codice (utile per l'impiego di formule matematiche). La maggioranza dei VPL è basata sul concetto di "boxes and arrows", dove le "box" (ovvero i rettangoli, circonferenze o simili) sono concepite come entità connesse tra di loro da "arrows" (frecce), linee o archi che rappresentano le relazioni. Solitamente l'ambiente per la programmazione visuale fornisce fin da subito tutto il necessario per poter "disegnare" un programma, mettendo a disposizione elementi grafici o icone manipolabili dagli utenti in maniera interattiva in base a una qualche specifica grammatica spaziale per la costruzione di un programma e, in rapporto ai linguaggi scritti, le regole sintattiche sono praticamente inesistenti.

Un "visually transformed language" è un linguaggio non visuale a cui si sovrappone una rappresentazione visuale [6]. Al contrario i linguaggi visuali hanno un'espressione visuale intrinseca per la quale non vi è alcun evidente equivalente testuale. Un utile controesempio rispetto ai linguaggi di programmazione visuali è Microsoft Visual Studio. I linguaggi che esso comprende (Visual Basic, Visual C#, Visual J#, ecc.) non sono linguaggi di programmazione visuali, anche se sono spesso erroneamente considerati per tali. Tutti questi linguaggi sono testuali e non grafici. MS Visual Studio è un ambiente di programmazione visuale, ma non un linguaggio di programmazione visuale, da qui nasce la confusione.

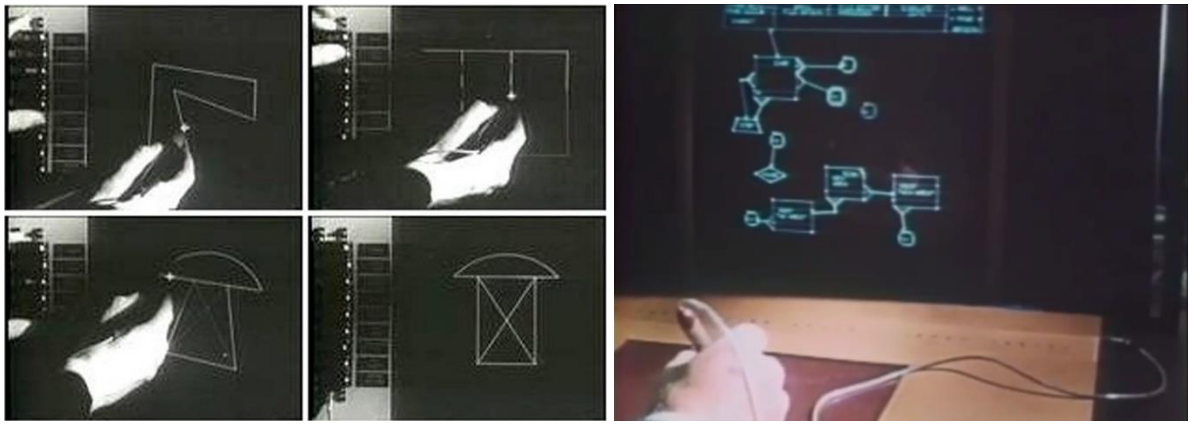


Figura 1. A sinistra l'interfaccia grafica di Sketchpad (1963) e a destra GraIL (1968). (fonte: blog.interfacevision.com)

Il primo VPL si può far risalire a Sketchpad del 1963, che fu il primo programma ad utilizzare un'interfaccia grafica per CAD (Computer Aided Crafting), mentre GraIL (Graphical Input Language) del 1968 fu il primo programma a icone (Fig.1). In Sketchpad, realizzato per il computer TX-2 al MIT, l'utente poteva lavorare con una penna ottica sul monitor per creare grafica 2D attraverso semplici primitive quali linee e cerchi, applicando operazioni come la copia e vincoli alla geometria delle forme disegnate. Nei decenni successivi i linguaggi visuali hanno subito una lenta ma costante diffusione, anche grazie all'evoluzione tecnologica e a una maggiore potenza di calcolo che permise un sempre più avanzato approccio grafico. La programmazione visuale è tuttora in continua evoluzione. L'idea resta convincente e in alcuni ambiti i VPL si sono ritagliati uno spazio rilevante: per esempio, uno dei campi in cui i VPL hanno riscosso particolare successo è quello dell'insegnamento, siccome permettono di spiegare più intuitivamente agli utenti come realizzare un programma (Fig.2). L'obiettivo più comune tra i VPL è quello di rendere la programmazione maggiormente accessibile, in particolare ridurre le frequenti difficoltà che i principianti si ritrovano ad affrontare quando iniziano a programmare. Gli attuali sviluppi in materia cercano di integrare l'approccio della programmazione visuale con i linguaggi di programmazione dataflow, in modo da disporre di un accesso immediato allo stato del programma, con conseguente debugging on-line, o la generazione e documentazione automatica del programma. Inoltre i linguaggi dataflow permettono anche la parallelizzazione automatica, che è probabilmente destinata a diventare una delle più grandi sfide di programmazione del futuro.

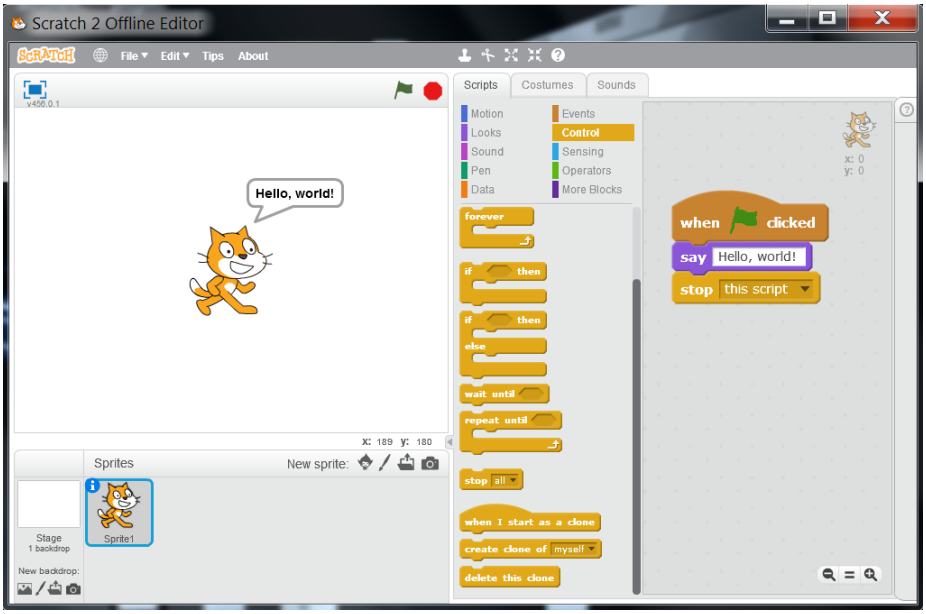


Figura 2. Implementazione di un programma "Hello, world!" nel linguaggio di programmazione Scratch.

2.1.1 Analisi di pro e contro dei VPL

I vantaggi dati dalla programmazione visuale sono la facilità di apprendimento e la possibilità di visualizzare lo stato del programma durante le fasi di debug. Inoltre la programmazione parallela, se gestita dal software, diventa quasi "istintiva" e soprattutto eseguita in automatico. Un altro aspetto molto importante è che la programmazione visuale può rendere più semplice la visione del quadro generale, in quanto il programma è definito dalla sua forma. Sia i principianti che gli utenti avanzati possono farsi un'idea di ciò che viene svolto dal programma semplicemente con un colpo d'occhio. In aggiunta, forme e colori permettono di rendere il codice più leggibile, avvantaggiando le abilità visuali dell'utente molto più di quanto siano in grado di fare i rientri (indentazioni) e l'uso dei colori. Tutto questo concorre a descrivere il programma più facilmente: si può presentarlo ad altri spiegandone i suoi vari punti, anche a persone che non hanno familiarità con il VPL che è stato utilizzato, mostrando loro direttamente il codice sorgente grafico, anziché dover disegnare un ulteriore diagramma astratto. Questo facilita anche la manutenzione, dato che anche se si osserva un codice VPL che non è familiare, si è in grado di vedere più agevolmente ciò che fa e come gli elementi si fondano insieme per creare il programma. Una struttura a blocchi inoltre sottintende che è spesso più semplice riorganizzare il codice grafico, al contrario di ciò che avviene con del codice testuale.
Un rilevante ambito in cui i VPL sono d'aiuto è la comprensione della sintassi di un linguaggio. Infatti, anche se nel codice si utilizzano parole valide, solo una piccola porzione di tutto quello che si può scrivere in un comune file di testo equivale a del codice valido per un dato linguaggio, e un'ancor più ridotta parte di esso genera un programma in grado di produrre un qualche output [7]. Invece, nei VPL gli elementi base di programmazione sono generalmente rappresentati da blocchi che possiedono indicazioni visive di come possono essere usati e collegati tra di loro. Così, per creare un programma, l'utente si ritrova a dover assemblare i blocchi come se fossero dei mattoncini di un gioco di costruzioni (come i Lego) o dei componenti elettronici, sistemandoli in modo da "incastrarsi" tra di loro. Di conseguenza risulta praticamente impossibile assemblare delle espressioni invalide. Un altro fattore importante è che i suddetti blocchi sono generalmente presentati con un catalogo principale di quelli esistenti, in modo che l'utente non debba leggere la documentazione o indovinare quali siano le funzioni a sua disposizione. Questi due punti rimuovono praticamente i più importanti problemi su cui i principianti si scontrano quando devono imparare la sintassi di un linguaggio di programmazione, permettendo loro di concentrarsi sulla logica del loro programma. Spesso, all'ambiente di sviluppo dei VPL è affiancato anche un ambiente di esecuzione semplificato, così gli utenti possono velocemente e facilmente eseguire i loro programmi appena creati e vedere i risultati. Questo non è necessariamente collegato in maniera diretta ai VPL, ma è un ulteriore fattore importante nel rendere più semplice ai principianti l'ottenere il primo programma in esecuzione, senza doversi preoccupare degli aspetti

tecnici. Questi punti non sono esclusivi ai VPL. Anche linguaggi di programmazione più astratti o semplificati ed IDE (Integrated Development Environment) hanno utilizzato idee simili al fine di semplificare l'apprendimento della sintassi, ma questa è un'area in cui le specificità dei VPL li fanno risaltare.

Nonostante i vantaggi elencati, finora i VPL non si avvicinano in alcun modo alla popolarità dei classici linguaggi di programmazione. Piuttosto, a volte sono percepiti come una delusione, in quanto la programmazione visuale rimane sempre programmazione.

Uno dei motivi è il fatto che al posto di disegnare a mano "boxes and arrows", essi richiedono ancora una definizione precisa e univoca del flusso di controlli. Anche se i VPL possono semplificare la programmazione e ridurre le difficoltà di sintassi, richiedono comunque agli utenti di familiarizzare con i concetti generali della programmazione e con quelli specifici del linguaggio. Quindi, per essere in grado di scrivere un programma con un VPL, si deve ancora ragionare come un programmatore. Esistono altri fattori che possono spiegare la cattiva reputazione che i VPL hanno tra molti sviluppatori. Se si considerano gli sviluppatori professionisti che scrivono programmi tutti i giorni, per loro la programmazione visuale sembra non aggiungere molto. Quando un utente familiare con la sintassi di un linguaggio di programmazione necessita di lavorare con molti componenti eterogenei o librerie, i vantaggi dei VPL per la comprensione della sintassi non sono molto interessanti.

In termini di rappresentazione delle espressioni, un programma testuale è al contempo molto compatto e aperto. Lo spazio grafico richiesto da un "blocco" di un VPL (restando leggibile e memorizzabile) permetterebbe di farci stare molte parole e si può facilmente aggiungere nuove parole, per estendere il linguaggio, facendo riferimento a funzioni esterne, mentre nei VPL il numero di forme e colori riconoscibili è limitato da quello che un utente può ricordare. Anche se alcuni VPL combinano tra loro forme e colori, le rappresentazioni visuali ottenibili non possono competere con la densità di informazioni del testo. Inoltre uno dei principali limiti con cui devono scontrarsi i VPL è il cosiddetto "limite di Deutsch" [8]. Si tratta di un aforisma riguardo alla densità di informazione di un VPL. Il termine venne coniato da Fred Lakin, in seguito a un intervento fatto da Laurence Peter Deutsch in un discorso sulla programmazione visuale di Scott Kim e Warren Robinett. Deutsch intervenne asserendo che "il problema con i linguaggi di programmazione visuali è che non si può avere più di 50 primitive visuali a schermo nello stesso tempo". Non è chiaro se un simile limite esista anche per i linguaggi di programmazione testuali e tale limite potrebbe anche essere superato applicando la modularità alla programmazione visiva, come avviene comunemente nella programmazione testuale. Tuttavia il limite di Deutsch è spesso citato come esempio tra i vantaggi dei linguaggi testuali rispetto a quelli visuali, evidenziando la maggior densità di informazione del testo.

Così, anche se i vantaggi dei VPL in termini di accessibilità e leggibilità del codice reggono ancora con parte degli sviluppatori, non sono un fattore di forza per la loro diffusione. Inoltre, gli ambienti di sviluppo VPL sono a volte specializzati, applicati a domini ridotti (per esempio il game design); in

questo modo i programmi possono essere eseguiti direttamente in un ambiente di esecuzione integrato e i comunemente utilizzati blocchi logici a disposizione non sono eccessivamente numerosi da rendere la comprensione della sintassi difficile. Quest'ultima è una caratteristica difficile da ottenere con un VPL general-purpose. Un altro fattore è che anche alcuni VPL "seri" hanno mostrato carenze sul piano tecnico, in particolare riguardo la loro velocità di esecuzione.
Dopo aver analizzato i pro e contro dei VPL, si passa ora a tentare una catalogazione di quelli che sembrano i VPL attualmente più diffusi.

2.1.2 Categorie di VPL

Una possibile catalogazione dei vari linguaggi visuali potrebbe essere applicata in base all'ambito per i quali sono stati ideati. In tal caso la maggior parte dei VPL sarebbe distinguibile nelle seguenti classi principali: educativi, multimedia, videogame, simulazioni, automazione, data warehousing / business intelligence. Inoltre i VPL si possono classificare, in base a come rappresentano su schermo le funzioni, in linguaggi icon-based, linguaggi form-based o linguaggi a diagrammi. Nel progetto di tesi si è ritenuto più utile, ai fini di analisi, una classificazione basata sul genere di rappresentazione visuale adottata, distinguendo sei tipologie, ovvero: Scratch e suoi simili, flowcharts-inspired, programmazione del flusso di dati, macchine a stati finiti, alberi comportamentali (Behavior Tree) e regole basate su eventi. Si valuteranno i vantaggi ed i punti deboli in termini di espressività e facilità di utilizzo, fornendo degli esempi di implementazioni [9] per le varie categorie.

1. Scratch e simili

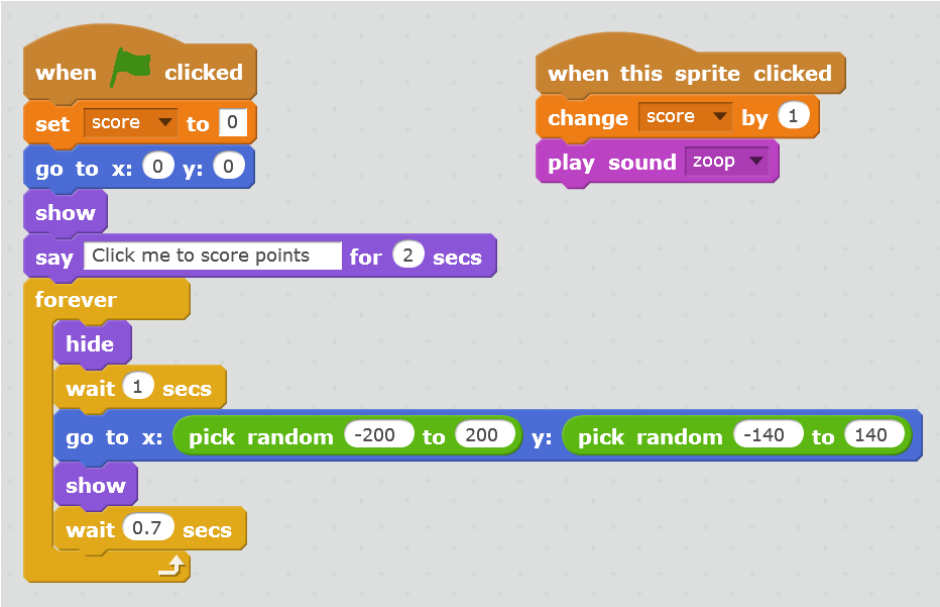


Figura 3. Esempi di sintassi in Scratch.

Quando si parla di VPL, oggi si tende a pensare subito a Scratch ed i suoi derivati (Fig.3). Non a caso, per realizzare il programma di tesi si è partiti analizzando in primis i linguaggi di programmazione di questa categoria, per la loro semplicità di utilizzo. Il rappresentante principe di questa categoria è Scratch appunto, a cui hanno fatto seguito vari altri linguaggi chiaramente ispirati al primo ma meno noti, il cui scopo spesso è quello educativo per avvicinare i più piccoli alla programmazione, alla robotica e simili, stimolando la loro creatività [10].

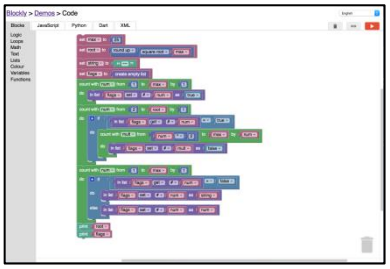
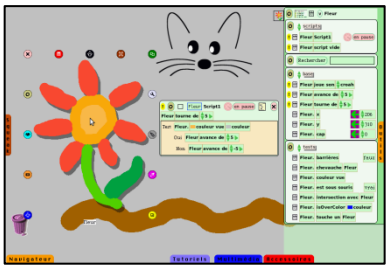
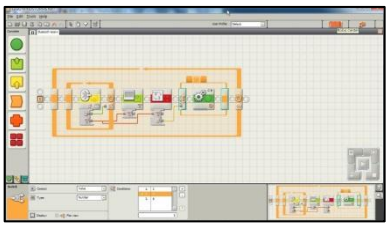
Rispetto ad altri VPL, questi linguaggi sono molto vicini al codice sorgente, la loro grammatica è in pratica essenzialmente la grammatica di un classico linguaggio di programmazione imperativo, ma grazie a degli indizi grafici si rende più evidente come combinare tra loro gli elementi base quali variabili, condizioni o controllori di flusso. In realtà il nucleo del linguaggio visuale può essere visto semplicemente come una guida sintattica per un linguaggio che assomiglia a C o Java.

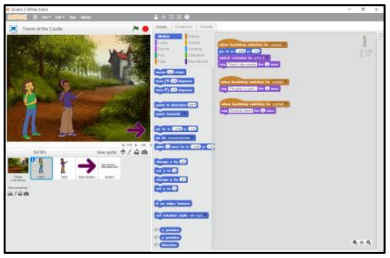
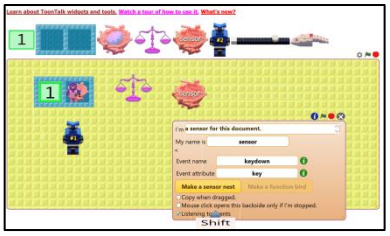
L'utilizzo di blocchi visuali aiuta la scoperta della sintassi e impedisce errori rendendo evidente come costruire un'istruzione funzionante e come i vari elementi si relazionino tra di loro. Facendo riferimento a Scratch (ma questi discorsi valgono anche per la quasi totalità degli altri simili), all'interno dei blocchi il testo viene usato per descrivere la funzione dei vari blocchi, mentre dei campi testuali sono usati per definire i nomi di variabili e valori.

È possibile estendere il linguaggio con blocchi personalizzati, che a loro volta sono definibili con Scratch. Oltre alla grammatica visuale, Scratch include un ambiente di esecuzione con un rendering grafico per creare facilmente animazioni di base o giochi. Ci sono utenti di Scratch che utilizzano

direttamente il formato di Scratch come si farebbe con un linguaggio di scripting, così come i suoi derivati che condividono gli stessi princìpi.

Il fatto che questi linguaggi di programmazione siano molto colorati e possiedano un'interfaccia grafica accattivante li rendono probabilmente più interessanti per quanti sarebbero altrimenti disorientati da un "muro di testo" e per tutti questi motivi Scratch ha trovato ampia diffusione tra le scuole per avvicinare i più giovani a muovere i primi passi nella programmazione [11].

EDUCATIVI	**Blockly**		Tool di Google per sviluppatori di applicazioni e non per bambini, ma noto grazie a molte applicazioni educative che usano Blockly per introdurre i bambini alla programmazione, come Blockly Games, App Inventor, Code.org, OzoBlockly, Open Roberta, Gameblox e altri [12]. Molto simile a Scratch, ma permette una facile esportazione in altri linguaggi quali Java, Python e XML.
EDUCATIVI	**Etoys**		Ambiente di creazione di progetti "multimediali" per uso educativo tra i bambini e gratuito [13]; Nato da Squeak, Etoys è alla base dell'ambiente di programmazione Scratch.
EDUCATIVI	**HopScotch**		VPL per dispositivi mobili touchscreen (disponibile su iPad) [14]. Permette a programmatori principianti o giovani di sviluppare semplici giochi. Ispirato a Scratch, ma ancora più semplificato.
SIMULAZIONI	**Lego Mindstorms NXT**		Creato da MIT Media Lab e LEGO (NXT-G è un VPL per il kit di robotica Lego Mindstorms NXT) [15]. Abbastanza complesso, ma con curva di apprendimento dolce, permette all'utente di controllare robot con il giroscopio dello smartphone.

SIMULAZIONI	**Minibloq**		Linguaggio di programmazione visuale per la robotica e schede Arduino compatibili, usato per insegnare la programmazione nelle scuole; Ha meno testo nella rappresentazione grafica rispetto a Scratch e simili, ma è affiancato al codice, il quale viene generato in tempo reale quando si posizionano i blocchi; ogni blocco è configurato in XML [16].
EDUCATIVI	**Scratch**		Permette l'insegnamento della programmazione tra gli studenti giovani [17]. Per realizzare storie animate, semplici giochi, simulazioni, arte interattiva e musica.
EDUCATIVI	**ToonTalk**		Sistema di programmazione per bambini [18]. Un programma è una sequenza di regole, ognuna con una testa e una coda; ogni regola appare come un robot, un programma come un team di robot.
EDUCATIVI	**TouchDevelop**		Un ambiente di programmazione cross-platform sviluppato da Microsoft Research [19]. Per creare app per dispositivi mobili come smartphone, tablet e laptop.

Sempre in questa categoria si segnalano: Amici, App Inventor for Android, Ardublock, Babuino, Bitbloq, graspIO, Lava, Mama, Modkit, Open Roberta, Snap! [20], StarLogo, Stencyl, TREPL, Tynker, Wyliodrin e molti altri.

2. Flowcharts-inspired

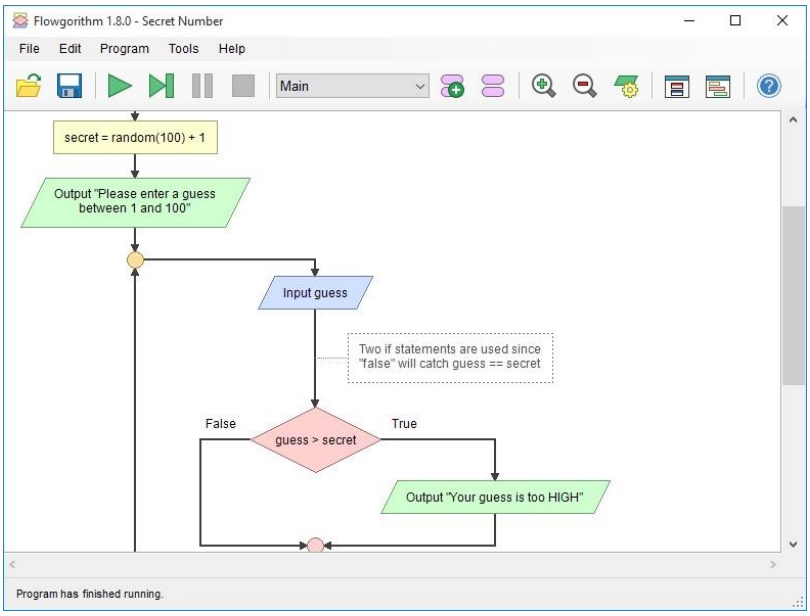
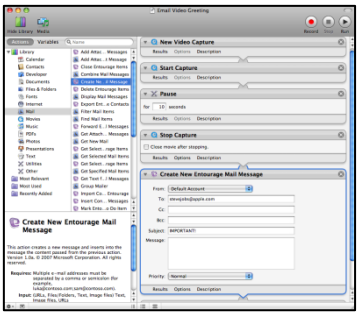
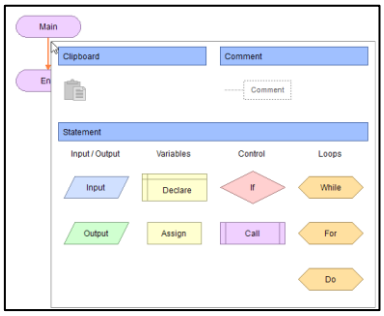
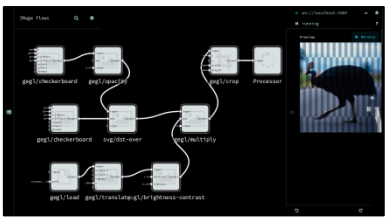
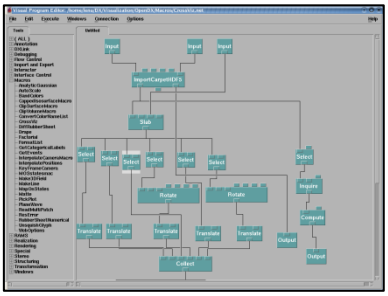
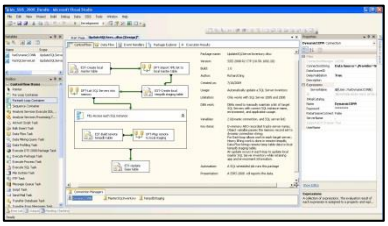
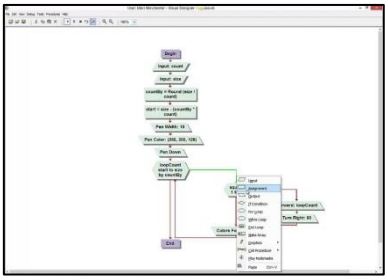


Figura 4. Flowgorithm.

Questa categoria di VPL condivide con la precedente l'idea di usare "boxes and arrows" per descrivere i programmi, ma si differenzia per l'utilizzo di una rappresentazione visuale vicina a quella dei diagrammi di flusso (flowchart) per descrivere il flusso di controllo principale (Fig.4). In questo caso si ha una sequenza diretta di esecuzione tra i blocchi, dove il flusso passa da un blocco a quello successivo, spesso con "biforcazioni" in cui il risultato in uscita di un blocco è utilizzato per scegliere quale blocco eseguire dopo.

Questa attenzione per l'esecuzione e la grammatica visuale minimale permettono di avere un formato semplice da comprendere e la scoperta della sintassi non è un problema. Lo svantaggio è che i costrutti logici creabili direttamente attraverso questo VPL sono limitati, siccome molto dipende da ciò che c'è dentro ai blocchi e che spesso è predeterminato e non modificabile dall'interfaccia grafica.

Esistono alcuni formati simili ai diagrammi di flusso, ma con una grammatica più estesa che permette di scrivere istruzioni più complesse direttamente dall'interfaccia grafica. Tra tali formati, vanno menzionati: programmazione del flusso di dati (dataflow programming), macchine a stati finiti (FSM) e gli alberi comportamentali (Behavior Tree).

AUTOMAZIONE	**Automator**		Applicazione sviluppata da Apple che permette di utilizzare il linguaggio di scripting AppleScript senza scrivere il codice, ma creando un diagramma di flusso scegliendo le varie azioni da fare eseguire allo script, come ruotare un'immagine o salvare un file.
EDUCATIVI	**Flowgorithm**		Crea diagrammi di flusso eseguibili che possono essere convertiti in diversi linguaggi, quali C++, C#, Delphi, Java, JavaScript, Perl, Python, Qbasic, Visual Basic .NET [21]. Il codice ricavato mantiene i colori del diagramma.
SIMULAZIONI	**Flow Hub & NoFlo**		Progetto nato da Kickstarter per ridurre il gap tra progettazione e sviluppo. Per aiutare a organizzare le proprie applicazioni, che sono create da componenti connessi in un grafo, in modo da scomporre così un problema in varie aree logiche [22].
SIMULAZIONI	**OpenDX**		Visualizzazione di dati scientifici utilizzando un linguaggio di programmazione visuale e un modello a flusso di dati [23].
DATA WAREHOUSING / BUSINESS INTELLIGENCE	**Sql Server Integration Services**		Integration Services è una piattaforma per costruire applicazioni di integrazione dati e di workflow [24]. Il suo campo di applicazione principale è il data warehousing perché fornisce tool per l'estrazione e trasformazione dei dati.
EDUCATIVI	**Visual Logic**		Permette agli studenti di programmare creando diagrammi di flusso eseguibili, per le piattaforme Windows [25]. Utilizzato di solito per insegnare agli studenti concetti introduttivi di programmazione.

Sempre in questa categoria si segnalano: Bonita BPM (e altri tool BPMN), WebML, Discovery Machine, Flowcode, Golaem Crowd's behavior editor, Grafcet, LARP, LlamaLab Automate, Node-RED, Raptor, tray.io e molti altri.

3. Programmazione del flusso di dati

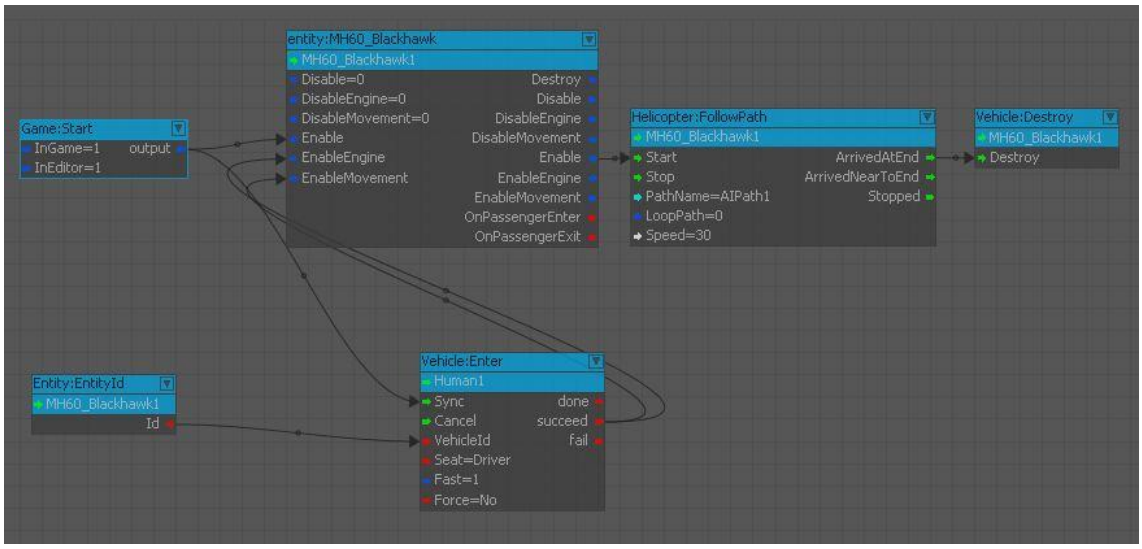
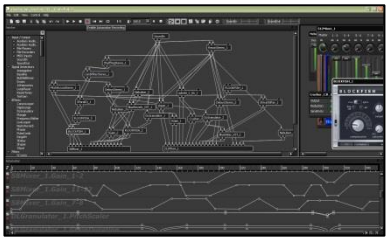
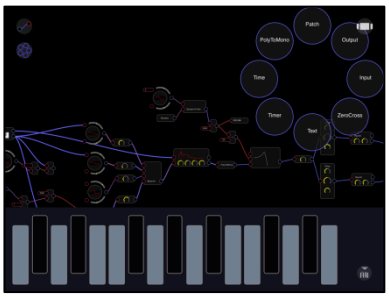
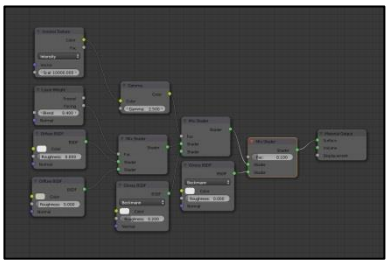
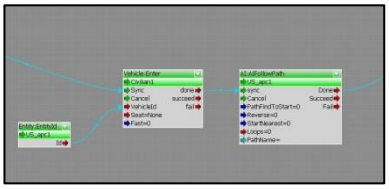
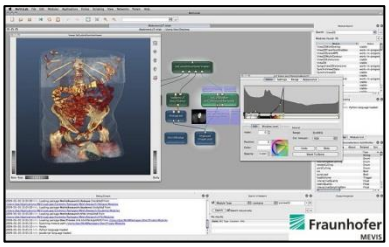
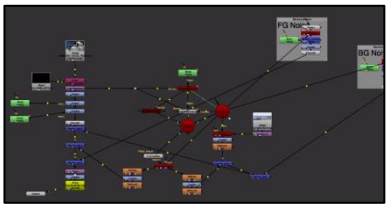
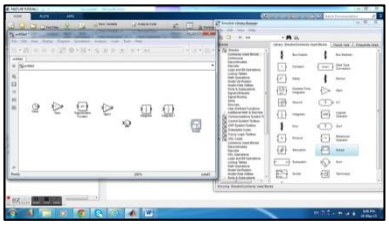


Figura 5. CryEngine Flow Graph.

Questo formato è probabilmente il VPL più utilizzato per applicazioni professionali, rivolto principalmente ai designer piuttosto che agli utenti finali o ai programmatori principianti.
In questo caso, i blocchi rappresentano delle funzioni e i loro ingressi e uscite sono collegati tra loro lungo il flusso di dati (Fig.5). Dunque, il flusso di esecuzione di un programma è dato dai collegamenti presenti tra l'output di un blocco e l'input di quello successivo. Si ha il vantaggio che la grammatica visuale è abbastanza semplice e di solito ci si deve limitare a scegliere quali blocchi si desidera utilizzare e come collegarli tra loro, perciò buona parte di ciò che definisce il programma è presente dentro i blocchi, che di solito sono definiti con un classico linguaggio di programmazione.
Anche se il formato può essere semplice, il problema è che necessita di blocchi esistenti e per questo motivo si adatta soprattutto ad applicazioni in cui i componenti logici di alto livello sono spesso gli stessi. Inoltre, richiede da parte degli utenti una comprensione di nozioni di programmazione correlate al flusso di dati, perciò spesso si rivolge a utenti che possiedono un pensiero tecnico, in grado di porre l'enfasi sul mostrare la struttura generale del programma e il flusso dati intorno ai blocchi aventi un comportamento precostituito.

MULTIMEDIA	**AudioMulch**		Un ambiente di creazione di musica e suono basato sul flusso del segnale audio [26]. L'interfaccia grafica utente è in stile "patch panel", in cui i moduli chiamati "contraption" possono essere collegati tra loro per instradare l'audio e processare i suoni.
MULTIMEDIA	**Audulus**		App per processare musica modulare con facilità (per Mac, iPhone, iPad) [27]. Permette di progettare sintetizzatori, nuovi suoni, elaborare audio; è pensato per un'interfaccia touch.
MULTIMEDIA	**Blender**		Software di modellazione, rigging, animazione, compositing e rendering di grafica 3D [28]. Include un cosiddetto "node editor".
VIDEOGIOCHI	**CryEngine**		Motore grafico sviluppato da Crytek per videogiochi; include l'editor Flow Graph che permette agli utenti di non dover avere conoscenze di programmazione [29].
SIMULAZIONI	**MeVisLab**		Framework per applicazioni cross-platform per elaborazione di immagini mediche e visualizzazione scientifica [30].
MULTIMEDIA	**Nuke**		Un linguaggio di programmazione visuale Python-based per il compositing di effetti visivi della The Foundry [31]. Usato per post-produzione professionale nei più importanti film e serie TV.
SIMULAZIONI	**Simulink**		Un software per la modellazione, simulazione e analisi di sistemi dinamici, sviluppato dalla compagnia statunitense MathWorks [32]. Questo software è strettamente integrato con MATLAB.

Sempre in questa categoria si segnalano: Aldebaran Choregraphe, Algo Design Lab, Cameleon, DaVinci Resolve, DRAKON, Filter Forge, Godot, Grasshopper 3D, Houdini, KNIME, Kyma, LabVIEW, Max (Max/MSP), Microsoft Visual Programming Language, NETLab Toolkit, Open Music, OpenWire,

Orange, Pure Data (Pd), Quartz Composer, Reaktor, RTMaps, Softimage ICE, SynthEdit, Tersus, Virtools, VVVV, OpenRTM, Spirops AI, Unreal Engine (Blueprints) e molti altri.

4. Macchine a stati finiti

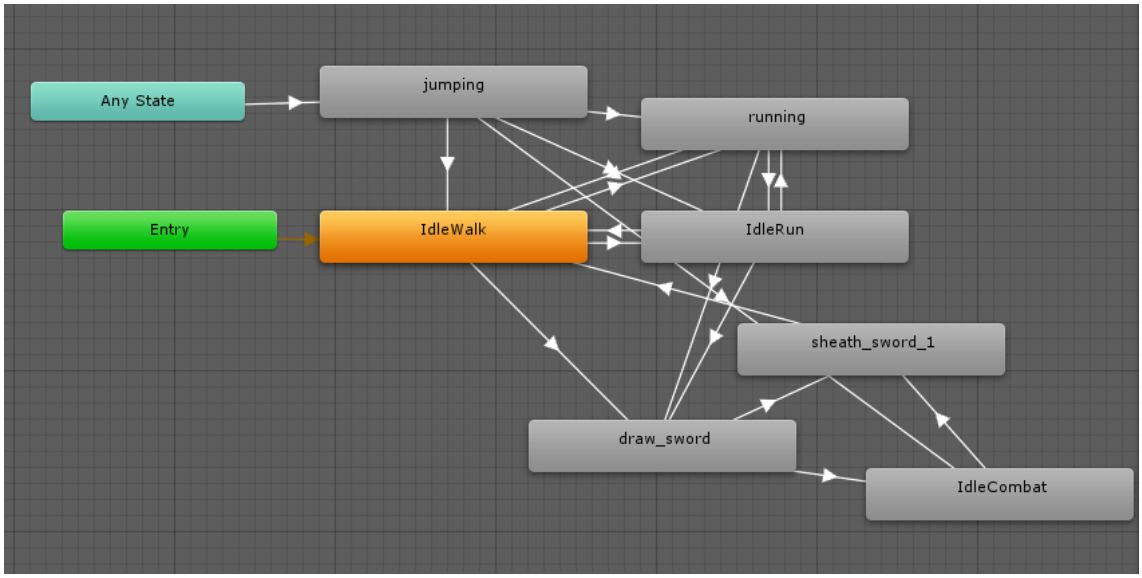
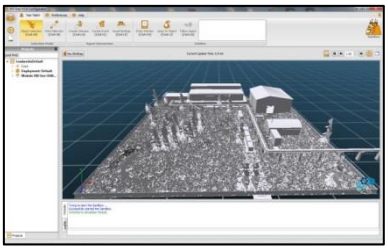
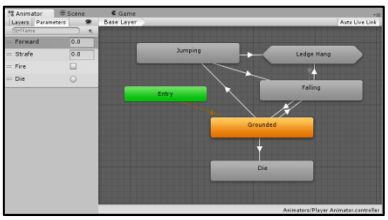


Figura 6. Unity 3D Animator Controller.

Le macchine a stati finiti (Final State Machine, FSM) definiscono stati e condizioni che attivano la transizione tra di essi e durante il cambio di stato si eseguono le istruzioni desiderate. Nella rappresentazione grafica delle FSM i blocchi corrispondono agli stati e i collegamenti tra essi corrispondono alle transizioni (Fig.6). Dunque l'utente progetta il flusso di esecuzione come con i precedenti diagrammi di flusso, collegando nel modo desiderato gli stati e definendo quali siano gli eventi che scatenano i cambiamenti di stato.

Il fatto che la logica della scelta degli stati successivi sia controllata direttamente dalle condizioni che attivano i cambiamenti di stato, piuttosto che dentro i blocchi, permette all'utente di creare istruzioni più complesse, nonostante si continuino ad utilizzare i blocchi. Lo svantaggio è che questo richiede all'utente di saper comprendere e gestire opportunamente le condizioni che scatenano i cambiamenti di stato, generalmente attraverso espressioni di testo. Per questo motivo, questo formato è più complesso da apprendere e con più testo, ma anche più espressivo dei precedenti casi della programmazione del flusso di dati e dei diagrammi di flusso. La creazione di programmi richiede una maggiore comprensione tecnica, conservando comunque una vista del quadro generale della struttura del programma e una grammatica visuale molto semplice.

VIDEOGIOCHI	**EKI One**		Middleware usato in Unity3D per la gestione dell'intelligenza artificiale.
VIDEOGIOCHI	**Unity 3D Animator Controller**		Software per la creazione di videogiochi 3D che include una visualizzazione a macchina a stati, che determina quali animazioni sono correntemente eseguite e le connette in modo uniforme [33].
VIDEOGIOCHI	**xaitControl**		Software per l'intelligenza artificiale e modelli comportamentali dei personaggi nei giochi e simulazioni. Include un debugger real-time.

Tra i programmi che permettono un'impostazione secondo questa categoria di macchine a stati finiti si segnala anche NodeCanvas, ma si ha scelto di includerlo principalmente nella categoria degli alberi comportamentali.

5. Alberi comportamentali (Behavior Tree)

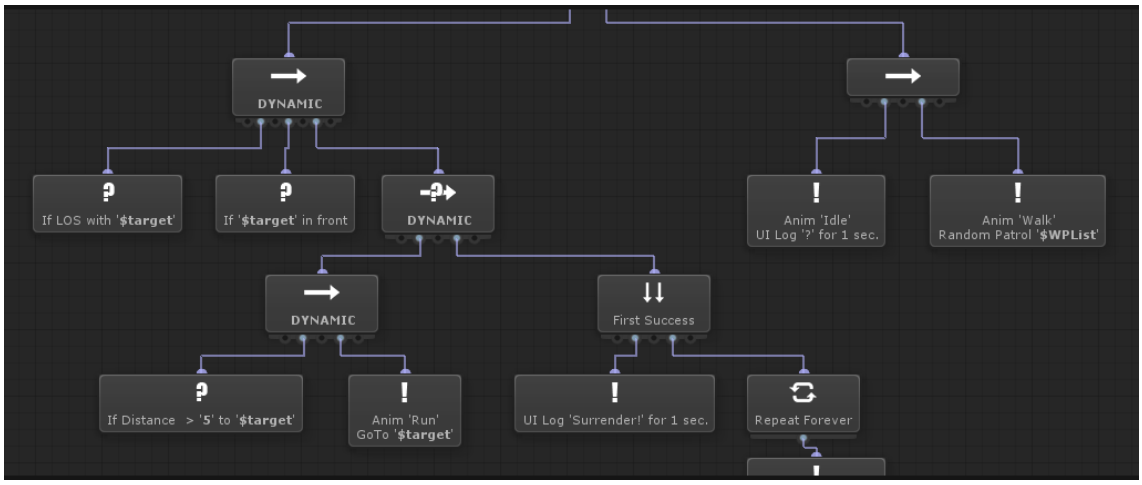


Figura 7. NodeCanvas.

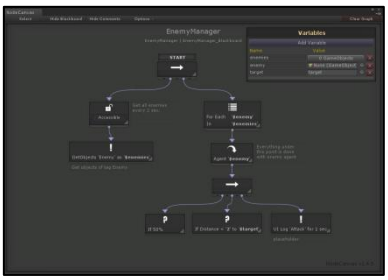
Gli alberi comportamentali (Behavior Tree, BT) si discostano leggermente dalle FSM. Negli ultimi tempi si è assistito a un crescente interesse verso i BT e molti videogiochi moderni ne fanno uso, ad

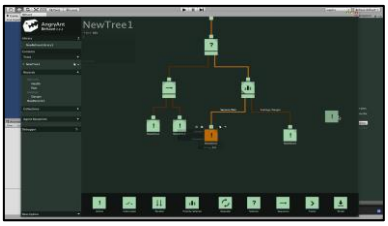
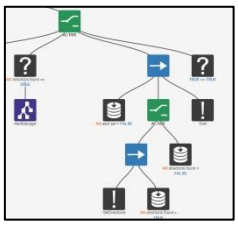
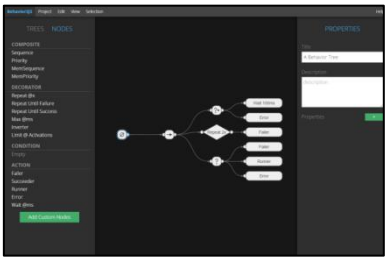
esempio per modellare l'intelligenza artificiale dei personaggi non giocanti (PNG). I comportamenti sono modellati come alberi e spesso per mezzo di editor grafici (Fig.7).

Come negli altri formati ispirati ai diagrammi di flusso, anche nei BT la grammatica visuale è composta da blocchi e collegamenti tra essi, definendo un flusso di esecuzione. La differenza in questo caso è che i blocchi vengono testati e ritornano indietro al precedente blocco uno stato di successo (o meno) come risposta, lungo una struttura ad albero composta da padri e figli. Questo permette di definire in un blocco un comportamento che dipende da quello dei suoi figli, controllando di conseguenza il flusso di esecuzione, come per esempio un blocco "sequenza" che testa tutti i suoi figli in ordine e si arresta se uno di essi fallisce.

Il vantaggio di un tale formato è che l'utente può definire la logica del programma direttamente dall'interfaccia grafica, senza dover aggiungere altro alla grammatica visuale e senza intervenire sul testo. Questa minore presenza della parte di logica nei campi testuali concorre a rendere il programma più leggibile rispetto agli altri formati e rispetto alle FSM si ha una modifica più diretta del programma dall'interfaccia grafica, ottenendo maggiori risultati dalla riorganizzazione dei collegamenti tra blocchi.

Se da un lato la semplicità della grammatica visuale rende la "scoperta" (l'apprendimento) della sintassi di base molto semplice, dall'altro lato i BT richiedono che l'utente comprenda come sono eseguiti gli alberi (concetto simile allo stack di esecuzione nella programmazione classica) e che familiarizzi con il comportamento dei blocchi per essere in grado di progettare dei programmi. Per questi motivi i BT hanno una curva di apprendimento più ripida rispetto a buona parte dei formati ispirati ai diagrammi di flusso, ma graficamente danno agli utenti un maggiore controllo. Propongono dunque un compromesso tra un formato come i diagrammi di flusso, aventi una grammatica visuale semplice e un'espressività dell'interfaccia limitata, e un formato come Scratch, con la complessità della programmazione classica e che fornisce un controllo a più basso livello.

VIDEOGIOCHI	**NodeCanvas**		Framework per Unity che permette di creare comportamenti di intelligenza artificiale e logica di gioco in un editor visuale intuitivo. Composto di due moduli: uno basato su alberi comportamentali e uno con macchine a stati finiti [34].
VIDEOGIOCHI	**RAIN, Rival Theory**		Motore per l'intelligenza artificiale utilizzabile in Unity [35]; Specializzato per gestire personaggi interattivi attraverso alberi comportamentali e macchine a stati finiti.

VIDEOGIOCHI	**AngryAnt Behave**		Plugin di Unity per progettare logiche comportamentali di intelligenza artificiale [36].
VIDEOGIOCHI	**craft ai**		Offre diversi servizi sulla base delle tecnologie di machine learning, con interesse sull'Internet of Things; fornisce AI-as-a-service comprendendo come l'utente interagisce con un sistema per adattare la risposta del sistema; apprende le regole direttamente dall'utente finale, senza che serva una persona che le scriva [37].
VIDEOGIOCHI	**Behavior3**		Fornisce un set di strumenti per creare e usare alberi comportamentali per giochi, simulazioni, robotica e altre applicazioni [38].

6. Regole basate su eventi

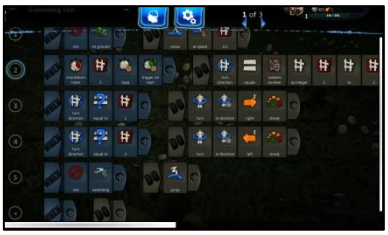
Figura 8. Esempi in Kodu.

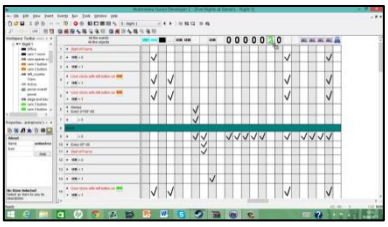
Di tutti i formati possibili di VPL questo è senza dubbio il più semplice e intuitivo. Si basa su un concetto di condizione-azione, dove l'utente definisce regole del tipo "se succede questo, fai quello", ovvero quando si verifica una certa condizione viene attivata una regola e si lanciano le relative

istruzioni. Graficamente si tende ad avere elementi abbastanza semplici, principalmente blocchi che possono essere utilizzati sia nell'ambito delle condizioni che tra le istruzioni, in modo da combinarli tra loro per definire le regole. In certi programmi è concesso pure combinare più blocchi tra loro per realizzare condizioni o istruzioni multiple. Un aspetto visuale solitamente molto schematico aiuta a riconoscere facilmente le regole esistenti a colpo d'occhio e a scoprire cosa può essere fatto.

La semplicità di questo formato si rivela sia il suo vantaggio che il suo limite: i programmi che si possono scrivere risultano abbastanza limitati e un linguaggio così ridotto può portare a sostenere che non si tratti più di programmazione, ma questo permette di renderlo facilmente comprensibile a qualsiasi persona, anche a chi non ha mai sviluppato un programma.

Di solito questa rappresentazione si rivolge agli utenti finali, permettendo loro di prendere il controllo e personalizzare le loro applicazioni con regole personalizzate. Il principale rappresentante di questa categoria è Kodu (Fig.8) a cui ha fatto seguito Project Spark; si parlerà in seguito di entrambi più nel dettaglio, analizzando il loro funzionamento.

EDUCATIVI	**AgentCubes / AgentSheets**		AgentCubes è un tool per la creazione di giochi e simulazioni 2D e 3D basati su agenti; utilizzato per l'insegnamento della programmazione agli studenti [39]. Evoluzione di AgentSheets.
VIDEOGIOCHI	**GameSalad**		Un tool visuale di creazione giochi sviluppato da GameSalad, Inc [40].
EDUCATIVI	**Kodu**		Tool di programmazione visuale per creare giochi 3D, basato sulle idee di Logo [41]. Usato come strumento per l'apprendimento educativo nelle scuole.
VIDEOGIOCHI	**Project Spark**		Videogioco che permette agli utenti di creare i propri giochi gestendone le logiche di interazione [42].

MULTIMEDIA	**Clickteam Fusion**		Clickteam Fusion e le versioni precedenti della Clickteam (come Multimedia Fusion, The Games Factory, Clic&Create, Klik&Play) permettono di creare semplici giochi 2D [43]. Si ha una tabella con una colonna per ogni oggetto di gioco e una riga per ogni condizione di azione che si intende specificare.

Sempre in questa categoria si segnalano: Construct 2, IFTTT, Netvibes DashBoard Of Things, Stagecast Creator e Zapier.

2.1.3 Conclusioni sui VPL

Anche se non si vede ancora un utilizzo diffuso professionale dei VPL, negli ultimi anni questi strumenti hanno guadagnato terreno in alcune aree, al punto che si può dire che sono finalmente diventati maturi. In queste aree sono strumenti efficienti adatto al loro scopo, siccome risultano effettivamente utili e mostrano vantaggi rispetto alla programmazione classica. In ambito educativo, per esempio, gli insegnanti utilizzano i VPL come un metodo più accessibile e accattivante per insegnare le basi della programmazione. Non solo Scratch e i suoi derivati che sono ampiamente usati, ma anche altri formati basati sui diagrammi di flusso sono stati utilizzati diffusamente nei giochi educativi.
Nell'ambito dei media ci sono settori, come il cinema e gli effetti speciali o la musica, in cui i designer sembrano utilizzare i VPL in maniera professionale, specialmente quelli che permettono loro di gestire la complessità grazie a blocchi riutilizzati in un formato leggibile. Per esempio i software 2D, 3D o di composizione musica spesso utilizzano VPL di programmazione di flusso di dati, i motori di animazione usano rappresentazioni di tipo FSM, mentre in ambito videoludico i designer di personaggi non giocanti (PNG) utilizzano vari tipi di VPL. In altre aree, i VPL sono rivolti più espressamente agli utenti finali, per personalizzare applicazioni o creare regole. In tal caso, spesso hanno un formato basato su regole scatenate da eventi, per la loro facilità di uso, ma non mancano i tentativi di utilizzare altri formati più espressivi, come nel caso delle interfacce per dashboard della Internet of Things o la personalizzazione dei dispositivi. Le applicazioni più creative, come per la robotica o la creazione di giochi, tendono a usare i VPL in modo più rivolto alla programmazione, con formati simili a Scratch che siano più accessibili rispetto ai classici linguaggi di programmazione.
Si può quindi affermare che i VPL siano sempre più utilizzati in contesti "seri" e gli utenti vedano un vero valore aggiunto nell'utilizzarli per le loro applicazioni, per merito della loro accessibilità e della loro capacità di aiutare a trattare la complessità di un sistema in modo più agevole.

2.2 Scratch

Scratch è un linguaggio di programmazione educativo gratuito progettato per i ragazzi delle scuole primarie e secondarie e i programmi del doposcuola, per insegnare loro la programmazione attraverso l'uso di primitive visive [44]. Viene utilizzato da studenti e insegnanti per creare storie interattive, animazioni e giochi, per la realizzazione di progetti con finalità pedagogiche e di intrattenimento, oltre a altri campi che interessano matematica, scienza, simulazioni, musica e arte. Il nome Scratch si ispira alla tecnica omonima dello scratch usata nel missaggio audio, visto che qui in modo analogo gli utenti possono remixare i progetti condivisi.

Nato nel 2003 e sviluppato dal Lifelong Kindergarten Group capitanato da Mitchel Resnick al MIT (Massachusetts Institute of Technology) Media Lab, si ispirò ad altri programmi quali Logo, Smalltalk, HyperCard, AgentSheets, Etoys e StarLogo. Nel 2007 introduce la possibilità agli utenti di condividere online i loro progetti e modificarli liberamente al motto di "Imagine, Program, Share" siccome, essendo a codice aperto, i progetti possono essere inviati direttamente dal programma al sito web di Scratch, e qualsiasi membro della comunità può scaricarne il codice per studiarlo o modificarlo in un nuovo progetto. I membri hanno anche la possibilità di creare gallerie di progetti, commentare, taggare e aggiungere ai propri preferiti i progetti di altri utenti. Nel 2013 riceve un rilevante aggiornamento alla versione Scratch 2.0 e viene reso disponibile sia sotto forma di software scaricabile e installabile per l'utilizzo offline sia direttamente sul sito ufficiale [17] come editor online. Esiste anche una comunità online per educatori denominata ScratchEd.

Il codice sorgente di Scratch è rilasciato con le licenze libere GPL2 e Scratch Source Code License. Scratch inizialmente era sviluppato in Smalltalk, mentre a partire dalla versione 2.0 è scritto in ActionScript, il linguaggio di Adobe Flash e di Adobe Air. I progetti sul sito vengono riprodotti su un browser, utilizzando Flash Player. Scratch oggi viene utilizzato nelle scuole di tutto il mondo per introdurre i bambini alla programmazione e stimolare la creatività, è considerato una pietra miliare nel suo settore e vanta innumerevoli derivazioni tra cui Snap! [20]. Attualmente conta circa 20 milioni di utenti registrati, prevalentemente giovani di 8-16 anni, per 22 milioni di progetti condivisi ed è in continua crescita.

Esiste infine anche una versione app pensata per tablet col nome ScratchJr, meno complessa e per i bambini. ScratchJr è una variante di Scratch sotto forma di app gratuita per iOS, Android e Chromebook. Si rivolge ai bambini di 5-7 anni, per insegnare loro a pensare in modo creativo e a ragionare sistematicamente, anche se non sono ancora in grado di leggere. Per questo motivo l'interfaccia utente è molto più semplice di Scratch (Fig.9), il numero di blocchi presenti è minore e soprattutto non contengono alcuna parola al loro interno, ma la loro funzione è rappresentata attraverso

icone. A differenza di Scratch, in questo caso i blocchi sono concatenati orizzontalmente da sinistra verso destra, come nella lettura [45].

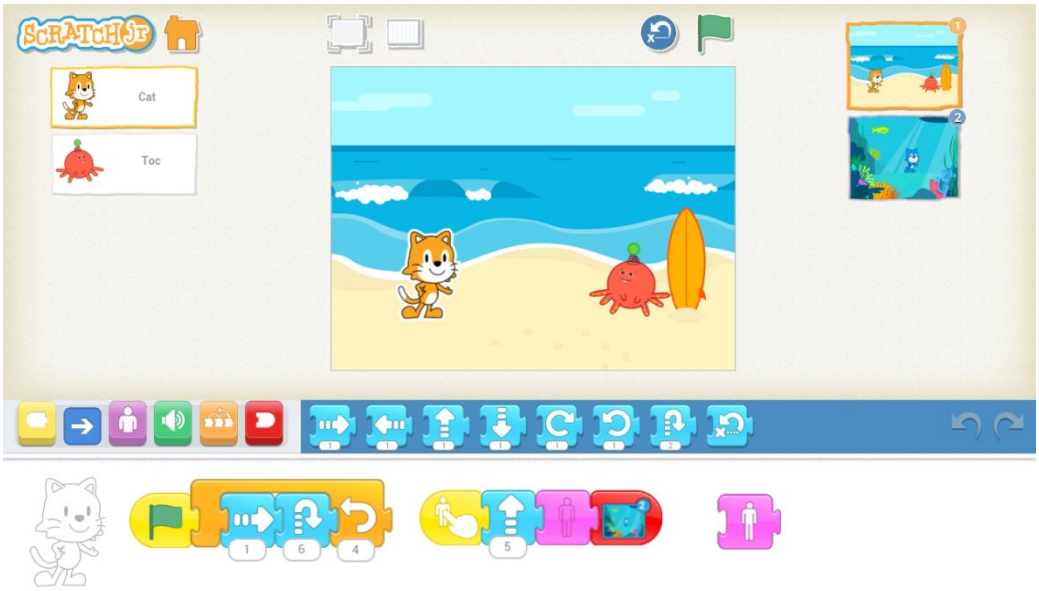


Figura 9. Interfaccia di ScratchJr.

2.2.1 Interfaccia utente

Scratch prevede un approccio orientato agli oggetti, con dei paradigmi di programmazione a eventi e programmazione imperativa. Gli oggetti grafici su cui agisce il codice sono denominati “sprite” e possono essere disegnati da zero nel semplice editor presente o importati dall’esterno. Il suo formato di programmazione si basa su blocchi grafici, creati per incastrarsi l’un l’altro come in un puzzle, ma solo se inseriti in una corretta successione, per evitare che si generino inesattezze nella sintassi. La sequenza di blocchi si distribuisce verticalmente, dall’alto verso il basso, come nella programmazione reale [46].

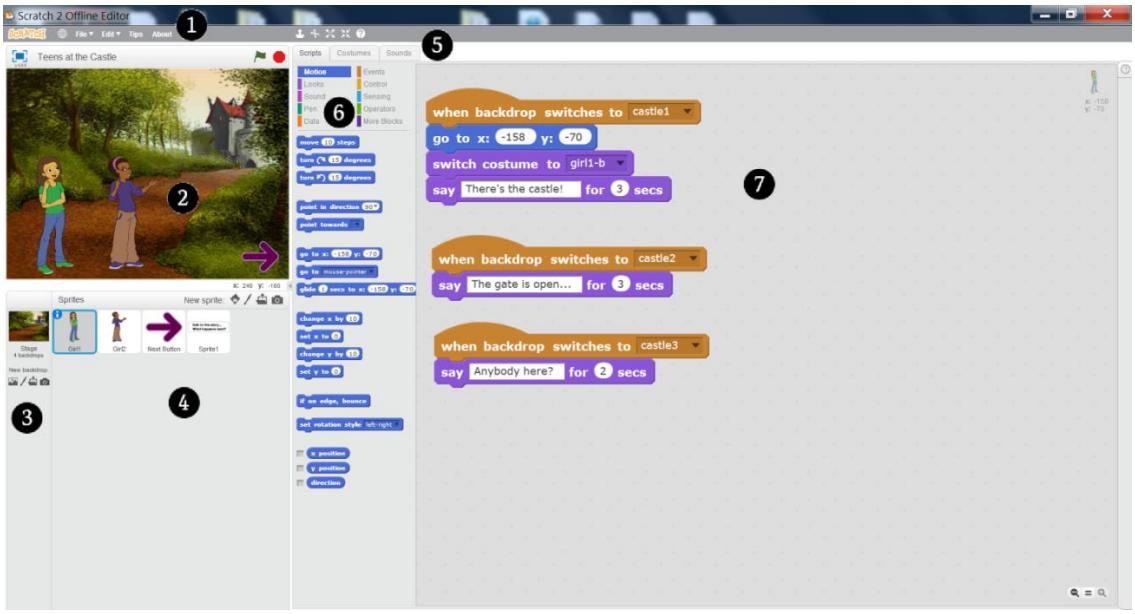


Figura 10. Interfaccia di Scratch.

L'interfaccia di Scratch è visibile in Figura 10 e si compone di diverse aree che si procede a illustrare.

1. **Menu**

   Menu con le principali operazioni quali aprire, salvare e condividere online un personale progetto.

2. **Stage area**

   Area in cui vengono mostrati gli sprite eseguire gli script assegnati loro. Qui si possono posizionare gli sprite nello stage e cliccando la bandierina verde si avviano gli script per vedere il risultato prodotto.

3. **Stage**

   Da qui si può selezionare lo stage e scegliere quale sfondo mostrare.

4. **Sprite area**

   Pannello in cui sono visualizzati tutti gli sprite esistenti nel progetto. Gli sprite posso essere aggiunti selezionandoli da una libreria di sprite preesistenti oppure disegnati a mano.

5. **Tab resource**

   Per lo sprite o lo stage selezionato nell'area degli sprite, si può decidere se visualizzare i suoi script, i costumi (che equivalgono a differenti rappresentazioni grafiche dello stesso, sfondi nel caso sia selezionato lo stage) o gli effetti audio associati a esso.

6. **Blocchi di script disponibili**

   Qui sono messi a disposizioni tutti i blocchi logici possibili tra cui scegliere, suddivisi per genere nelle categorie in alto, ognuna avente un diverso colore per renderle più facilmente distinguibili.

7. **Script area**

   In questa finestra vi sono tutti gli script generati per l'elemento attualmente selezionato.

I blocchi di codice selezionabili sono raggruppati per argomento nelle seguenti categorie:

- **Motion** – Muove gli Sprite e cambia gli angoli.
- **Events** – Blocchi di gestione degli eventi e da porre come testata.
- **Looks** – Controlla la visibilità, i costumi e l'output.
- **Control** – Istruzioni "if" e strutture cicliche.
- **Sound** – Esegue brani audio e sequenze audio programmabili.
- **Sensing** – Sensori per gli Sprite e input utente.
- **Pen** – Supporto al disegno e alla grafica.
- **Operators** – Operatori matematici e booleani.
- **Data** – Uso di variabili e assegnazione di valori.
- **More Blocks** – Per creare un blocco personalizzato riutilizzabile o per aggiungere estensioni come blocchi specializzati per LEGO WeDo o PicoBoard.

Selezionato uno sprite, per generare il suo script si selezionano i blocchi di codice desiderati trascinandoli nell'area del codice a destra. Un blocco può essere rimosso trascinandolo fuori da quest'area. Alcuni blocchi possiedono al loro interno un campo riempibile dall'utente, con numeri o testo, o selezionando un valore da una lista. Qualsiasi blocco può essere provato con un doppio clic in modo da vederne in anteprima l'azione.

I blocchi hanno una descrizione testuale al loro interno che ne definisce la funzione, la cui lingua può essere selezionata dai menu e i costrutti come "when" o "if-then-else" richiamano alla programmazione pura. Invece la forma dei blocchi ha il solo scopo di aiutare l'utente a capire come essi possano incastrarsi tra loro: per esempio una sequenza di blocchi deve sempre iniziare con un Event di tipo "when", che sono arrotondati nella parte superiore, mentre negli "if" ci si deve inserire un blocco Operator o Sensing che sia spigoloso nelle estremità laterali. In Figura 11 sono visibili alcuni script di esempio realizzati in Scratch.

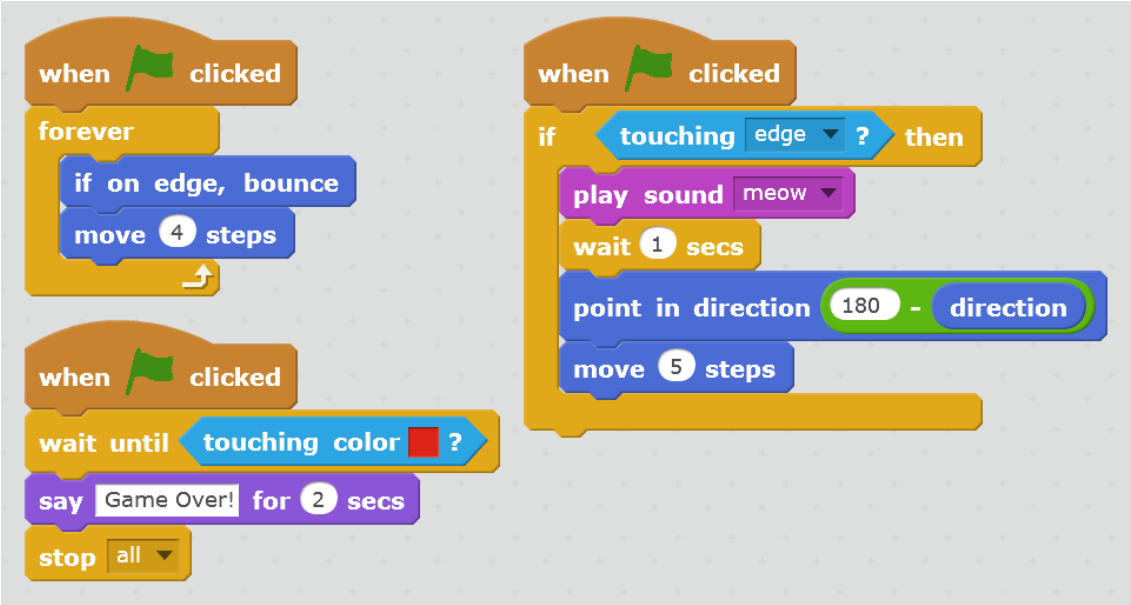


Figura 11. Esempi di script realizzabili.

Tra i limiti riscontrabili da Scratch si segnala l'impossibilità di inserire video e animazioni. Non si può riprodurre un video, ma è permesso caricare una GIF animata e farla funzionare scorrendo velocemente i vari costumi di uno sprite, in modo da simulare un'animazione.

Un altro problema rilevante è che Scratch non permette di gestire elementi 3D, quindi non è possibile ricreare nemmeno un semplice cubo. Inoltre durante l'esecuzione del proprio programma, gli sprite sono sempre spostabili di coordinate se si trascinano col mouse, evento che a volte può verificarsi per sbaglio quando si vorrebbe semplicemente cliccare su di essi.

Una mancanza fastidiosa che si è riscontrata è che non è possibile definire "scene di gioco" separate, dove poter inserire oggetti (sprite) diversi, come è possibile fare, ad esempio, in Blender. In Scratch, per ottenere un risultato analogo si deve ricorrere a un blocco Event di tipo "broadcast" che invia a tutti un messaggio di cambio scena e ogni oggetto, se lo riceve, modifica la sua visibilità o il posizionamento. Per ogni "scena" si deve indicare se l'oggetto è visibile o nascosto: se si dichiara visibile all'inizio, lo resta finché non si arriva a una scena successiva, in cui lo script lo setti nascosto e da quel momento rimarrà nascosto fino a quando non lo si dichiara nuovamente visibile. In contemporanea, all'arrivo di un messaggio in broadcast di cambio scena, è possibile per esempio cambiare lo sfondo.

2.2.2 Test

Per comprendere meglio nella pratica il funzionamento di Scratch, si è effettuato un semplice test, cercando di riprodurre risultati ottenibili, ad esempio, con Blender. Il test scelto per mettere alla prova il programma è stato dapprima realizzato in Blender. In questo modo sono venuti alla luce alcuni limiti intrinsechi del linguaggio grafico di Scratch.
Il test consiste delle seguenti tre scene:

- "Start", in cui ci sono tutti gli oggetti;
- "Scena Cubo", dove c'è solo l'oggetto cubo;
- "Scena Sfera", dove c'è solo l'oggetto sfera.

L'interazione che si desidera ricreare è la seguente:

- un clic su Suzanne produce una transizione a "Scena Sfera";
- un clic su Cubo produce una transizione a "Scena Cubo";
- una pressione della barra spaziatrice produce una transizione alla scena "Start", indipendentemente dalla scena correntemente impostata;
- un clic su Torus scatena l'inizio di un'animazione di Suzanne;
- una pressione del tasto "S" quando il puntatore è su Sfera causa la riproduzione di un effetto sonoro.

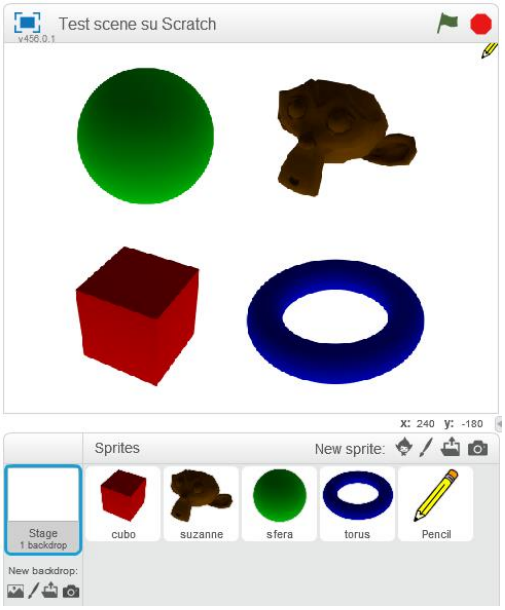


Figura 12. Scena del test.

Gli sprite impiegati sono immagini bidimensionali importate e corrispondenti agli oggetti elementari di Blender (Fig.12). Per realizzare il comportamento desiderato, la soluzione che si è ritenuta migliore da adottare è quella in cui si gestisce la scena Start attraverso script interni allo stage di sfondo. Lo script inserito nello stage si occupa di inviare il messaggio di "Start" in broadcast, sia all'avvio del programma che alla pressione della barra spaziatrice (Fig.13). Analogamente, negli script degli sprite richiesti ci saranno dei broadcast che gestiscono il passaggio alle altre due scene.

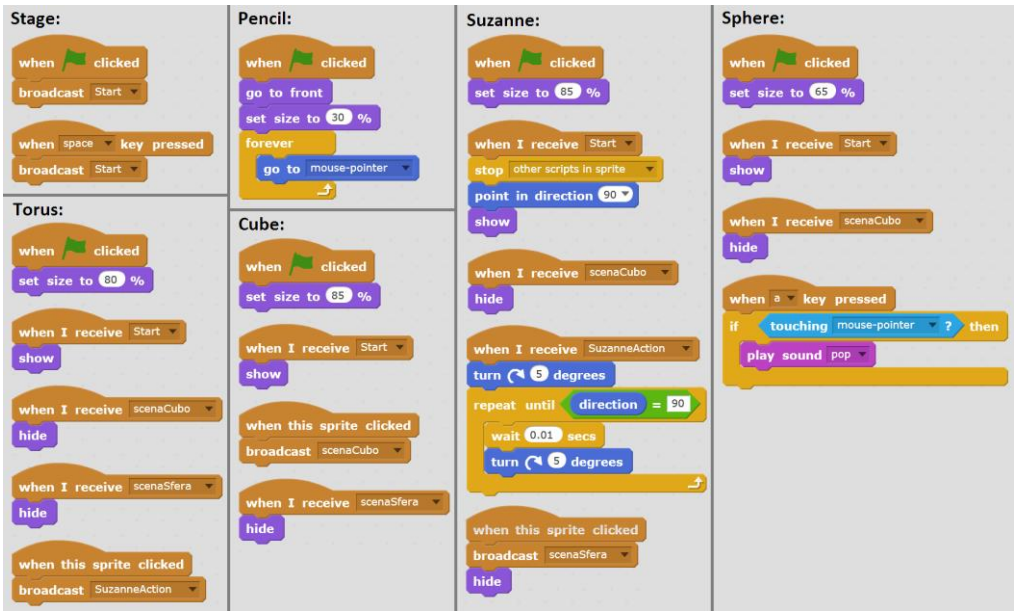


Figura 13. Raccolta degli script generati per il test.

Per tutti gli sprite rappresentanti degli oggetti, si devono inserire dei blocchi che gestiscano la visibilità o meno ("show" o "hide") di tali sprite quando si rileva un broadcast di passaggio a una certa scena.

Per Suzanne, quando si riceve un messaggio di broadcast indicante di tornare alla scena Start, si deve aggiungere un blocco di stop, che termini l'eventuale rotazione in corso che poteva avere questo oggetto, per successivamente concatenarci un blocco indicante di resettare la rotazione, oltre ad uno "show" che assicuri che l'oggetto sia visibile.

Affinché la sfera riproduca un effetto sonoro quando si preme un tasto su di essa, serve una combinazione di blocchi che controlli che il clic sia effettuato quando lo sprite del pointer sta toccando la sfera. Un problema nascosto è che questo controllo scatena l'audio anche se la sfera era attualmente invisibile in quella scena; un problema aggirabile se si complicasse la logica aggiungendo una variabile rappresentante la visibilità da poter essere testata nel blocco di "if".

Lo sprite pencil rappresenta il puntatore, ma in Scratch viene già visualizzato il puntatore del mouse e quindi non potrà sostituirlo, ma se lo si desidera è affiancabile ad esso. Viene forzato a essere

visualizzato davanti agli altri oggetti grazie all'iniziale blocco "go to front" e un blocco di controllo ciclico provvede infine a far sì che lo sprite segua sempre le coordinate del mouse.

2.3 Kodu

Kodu è un ambiente di sviluppo integrato per la programmazione [47] della Microsoft's FUture Social Experience (FUSE) Labs (letteralmente "Laboratorio per le esperienze sociali future di Microsoft"), un'azienda di ricerca appartenente a Microsoft. Il suo obiettivo è quello di permettere ad utenti inesperti di realizzare giochi interattivi in ambienti 3D. In origine noto come Boku, Kodu fu rilasciato al pubblico il 30 Giugno 2009 sull'Xbox Live Marketplace ed è disponibile per Xbox 360 e scaricabile gratuitamente per Windows, da Windows XP a Windows 10 [48]. Attualmente Kodu non riceve più aggiornamenti per la versione Xbox 360, ma la versione Windows continua a essere molto supportata dagli sviluppatori [41].

Kodu è uno strumento di programmazione visuale pensato specificatamente per lo sviluppo di giochi, progettato per essere accessibile ai bambini e divertente per chiunque [49]. Costruito sulle idee iniziate con Logo negli anni '60 e altri progetti correnti come AgentSheets, Squeak e Alice, Kodu, rispetto a questi altri progetti, permette di scrivere codice attraverso elementi visuali e i programmi sono eseguiti in un ambiente di simulazione 3D, simile a Alice. Kodu Game Lab è stato anche usato come strumento per l'apprendimento educativo in scuole selezionate e centri di apprendimento.

Il linguaggio di programmazione è semplice e completamente basato su icone. La semplicità di Kodu è ottenuta ponendo l'attività di programmazione in un ambiente di simulazione sostanzialmente completo e si può interagire usando un controller di gioco o una combinazione di mouse e tastiera. Fornisce primitive specializzate derivate da scenari di gioco. I programmi sono espressi ad alto livello, in termini fisici, utilizzando concetti quali visione, udito, colore, collisione e tempo, con il paradigma dei sensori costituito da un linguaggio basato su regole, in base alle condizioni e azioni.

L'utente programma i comportamenti dei personaggi in un mondo tridimensionale, in cui è possibile intervenire anche sul terreno, grazie a un editor flessibile con cui decidere l'aspetto del mondo di gioco e le dimensioni, con un'ampia varietà di superfici, texture e colori, oltre alla possibilità di inserire elementi quali ponti e percorsi. Sono messi a disposizione inoltre una ventina di personaggi diversi con differenti caratteristiche e altrettanti oggetti con cui poterli fare interagire.

Kodu permette di gestire istruzioni condizionali, punteggi, timer, suoni, effetti visivi e tutti quegli altri elementi necessari a costruire giochi interessanti. Si può controllare la camera, per esempio fissandola in una posizione specifica o, in alternativa, muovendola con il giocatore per una visuale in prima persona.

Gli oggetti possiedono proprietà configurabili che controllano la velocità, l'accelerazione, la vulnerabilità, i punti ferita, il danno dei missili e relativo tempo di ricarica, l'udito, la forza fisica e così via. Dunque, Kodu fornisce un giusto compromesso tra la ricchezza di funzionalità desiderata e la facilità di utilizzo. In questo modo, chiunque ha la capacità di realizzare diverse tipologie di videogioco, dai giochi di ruolo ai platform, puzzle, corse, strategia, FPS, avventura e altri. I giochi realizzati sono poi condivisibili via Xbox Live o su Planet Kodu.

A differenza di Scratch, in Kodu non è possibile importare personaggi o disegnarli da zero. Non è nemmeno permesso importare suoni, musiche o sfondi, come non si possono creare nuovi comandi o richiedere un input testuale al giocatore. Soprattutto, i comandi presenti sono rivolti principalmente alla realizzazione di giochi. Kodu scoraggia o addirittura preclude agli utenti la possibilità di creare storie animate, presentazioni o opere d'arte. Si possono realizzare storie e presentazioni, ma rispetto a Scratch esse rappresentano una percentuale nettamente inferiore dei progetti realizzati dagli utenti.

2.3.1 Interfaccia utente

Quando si inizia un nuovo progetto, l'utente si trova a disposizione un mondo virtuale vuoto e piatto. Nella parte inferiore dello schermo si ha un menu attraverso il quale poter fare le prime operazioni tra le quali avviare il gioco su cui si sta lavorando, aggiungere oggetti o spostare la camera, creare percorsi o modellare il terreno (Fig.14).

Figura 14. Schermata dell'editor generale di Kodu.

Una volta posizionato un oggetto si può accedere alle sue pagine di script cliccando su di esso.
Ogni oggetto possiede righe di script dette “regole”, che a loro volta sono suddivise in condizioni e azioni. La sintassi dei comandi di una regola è semplice e consistente: l’utente indica un evento e poi decide l’azione che ne deve risultare. Per definire una regola, l’utente deve selezionare i tasselli desiderati da un menu a comparsa.
Ogni riga corrisponde a una regola, che inizia con la parola chiave “When” seguita da un evento e, dopo di questo, la parola chiave “Do” seguita dal comportamento generato. Le righe sono numerate per indicare la priorità con cui si testano le regole. L’intero concetto dei When e Do è un qualcosa che si ripete in tutti i linguaggi di programmazione. Fin dalle origini della programmazione negli anni ‘50, il concetto di “when” e “do” diventa integrale della codifica. Li si è chiamati e combinati con altre parole come “if”, “while”, “else” e “for”, per nominarne alcuni. Questi comandi nel codice sono usati per dire cosa deve fare un sistema, come deve farlo e quando farlo. Kodu semplifica il tutto nelle due sezioni: “When: succede un certo evento”, “Do: una certa azione”. Per ogni riga si valuta il “When”, a quel punto, se la condizione è verificata, si esegue il “Do”.
La grammatica è: “When” <condizione> “Do” <azione>. Dove <condizione> è <sensore> [<filtro> …], mentre <azione> è <verbo> [<modificatore> …]. Un “When” include un primo tassello che ha la funzione di sensore e gli eventuali tasselli successivi ad esso presenti in questo “When” servono a specificare meglio l’evento, comportandosi da filtro. In modo analogo, un “Do” possiede un tassello iniziale che indica l’azione da eseguire, mentre gli eventuali tasselli successivi si comportano da modificatori.
Il classico “Hello, world!”, in Kodu si può far corrispondere a: “When: see – fruit; Do: move – towards”. Una variante più strutturata potrebbe essere per esempio: “When: see – red – fruit; Do: move – towards – quickly”.

Figura 15. Esempio di script realizzabili.

Un'impostazione così semplificata degli eventi permette anche a dei bambini di capire al volo i concetti espressi. Infatti, come visibile negli esempi in Figura 15, è sufficiente leggere una regola da sinistra verso destra per avere automaticamente una sorta di frase compiuta. Una grande varietà di opzioni e comandi rendono possibili molte ore di approfondimento e divertimento.

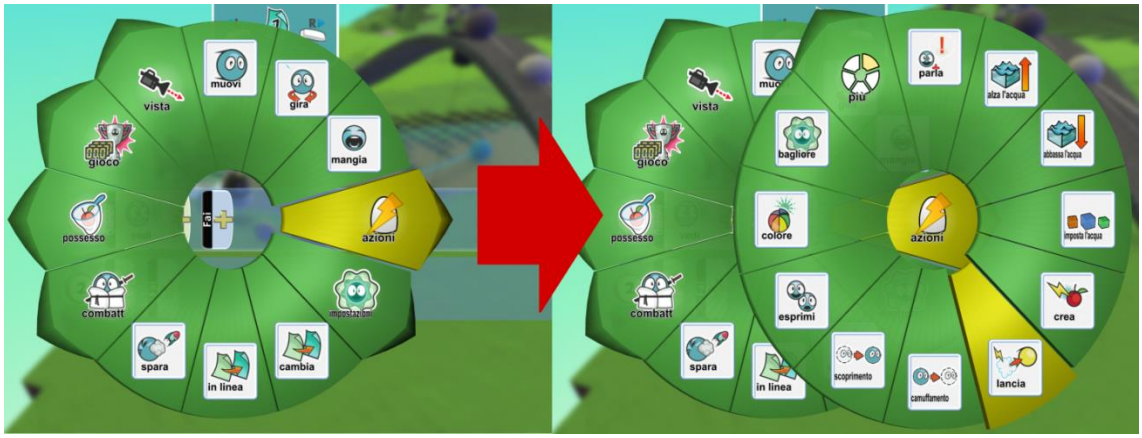


Figura 16. Esempio di navigazione del menu di selezione azione.

Per dichiarare un "When" o "Do" si clicca sul simbolo "+" al fondo, che fa apparire un menu circolare di selezione del tassello desiderato. Da questo primo menu circolare si ha a disposizione i principali tipi di condizioni o azioni fattibili e in base a ciò che si seleziona si può aprire un nuovo sottomenu con opzioni specifiche al genere scelto, come mostrato in Figura 16. Le funzioni a disposizione che compaiono variano se si tratta di un When o di un Do. Inoltre, se si ha selezionato un primo tassello e se ne vuole aggiungere un secondo che specifichi ulteriormente il primo, cliccando nuovamente sul "+" appariranno tasselli che differiscono in modo contestuale alla condizione o azione che si stava dichiarando. Per esempio, se nel When si inizia indicando un tassello "tastiera", provando ad aggiungere un secondo tassello ad esso compariranno tra cui scegliere solo quelli indicanti quale tasto della tastiera si vuole che sia premuto. In questo modo si semplifica enormemente la navigazione tra i tasselli da scegliere, evitando al contempo che l'utente possa generare regole non valide.

Si noti che un "When" può anche non possedere alcun tassello, in tale caso è sottointeso che sia sempre verificato e il suo "Do" sia sempre valido. Al contrario se si lascia un "Do" non dichiarato non succederà nulla al verificarsi di quell'evento.

Una importante funzionalità a disposizione è quella di realizzare condizioni multiple e azioni multiple. Se si vuole scatenare un'azione solo quando sono verificate due condizioni, si dichiara un primo "When" lasciandogli il "Do" vuoto e nella riga sottostante si dichiara nel nuovo "When" la seconda condizione con il "Do" contenente l'azione voluta. Per fare in modo che la seconda regola avvenga solo quando è verificata anche la prima, si deve trascinare la seconda regola a destra, in modo che risulti

annidata alla precedente (Fig.17), nello stesso modo in cui apparirebbero due costrutti “if” annidati nella programmazione pura.

Se invece si desidera scatenare due azioni diverse al verificarsi di un’unica condizione, nella prima regola si dichiara il When e nel Do si specifica una prima azione, mentre la regola successiva la si sposta a destra e si lascia il When vuoto (ovvero sarà sempre realizzata, ma solo se prima si verifica il When precedente) dichiarando nel Do la seconda azione voluta.

Figura 17. Esempio di script per un personaggio, con condizioni annidate.

L’ultimo importante concetto da comprendere è quello delle pagine. Ogni oggetto possiede un totale di 12 diverse pagine di programmazione. Se per esempio un nemico deve comportarsi diversamente in base alla fase di gioco in cui ci si trova, si può realizzare associando pagine diverse ai vari momenti: si definiscono le regole per la prima pagina e se si rileva un passaggio a una fase nuova si applica nel “Do” un passaggio alla pagina successiva. Simile al concetto di pagine, andando nelle impostazioni del mondo è possibile anche indicare un livello successivo, in modo che, quando si verifica una condizione desiderata, venga caricato un intero progetto di Kodu differente. Questo è utile quando si vuole realizzare un cambio di livello nel proprio gioco e il livello successivo deve contenere oggetti differenti da quello attuale. Il limite è che un livello (progetto di Kodu) può portare a caricare un singolo nuovo progetto, ovvero i livelli hanno solo un avanzamento sequenziale, impedendo di permettere al giocatore la possibilità di scegliere come proseguire nei livelli. In Kodu sono anche vietati collegamenti circolari tra i livelli, quindi se dal livello ‘A’ si passa al livello ‘B’, poi non è più possibile tornare al livello ‘A’, ma si potrà proseguire in ‘C’, da cui a sua volta non si potrà tornare circolarmente né ad ‘A’ né a ‘B’.

Un altro difetto di Kodu è che non è possibile memorizzare la posizione di un oggetto per ripristinarlo in quel posto successivamente. Si potrebbe voler creare un "Do" che imposta un punteggio nascosto a un certo valore indicante la fase di gioco a cui si sta passando e così ogni oggetto può reagire diversamente quando si passa a una certa fase, ma se un oggetto viene cancellato poi non si può ripristinare nello stesso posto. Una soluzione sarebbe quella di usare un "creabile", ovvero un oggetto pensato appositamente per rigenerare cloni di quel tipo di oggetto, in quel posto e con quella logica successivamente a una cancellazione di un precedente clone. Ma i creabili richiedono un oggetto definito creabile che risulta invisibile e un generatore di creabili che si può rendere invisibile o meno.
Tra gli altri limiti di Kodu si segnala che non permette di caricare oggetti personali oltre a quelli presenti all'interno del gioco e non ci sono oggetti piatti, solo tridimensionali, anche per questo motivo non si può contare su un editor interno per generarne di nuovi. Non è possibile aggiungere effetti audio personali, ma si deve usare quelli presenti nel gioco, che sono più di 100 tra effetti audio e musiche di sottofondo.
Si segnala infine la possibilità di generare un file di testo stampabile corrispondente al proprio progetto. Tale file include in formato testuale una lista degli oggetti e per ognuno elenca riga per riga le sue regole "When-Do" suddividendole per pagina.

2.3.2 Test

Si è provato a ricreare un progetto quanto più simile al test precedentemente realizzato per Scratch, in modo da valutare meglio le differenze tra questi due linguaggi.
In questo caso, si usano degli oggetti esistenti in Kodu, non potendo importare dei modelli esterni.

Figura 18. Scena del test in Kodu.

Visto che il progetto consiste in più scene in cui alcuni oggetti sono presenti e altri no a seconda della scena corrente, si deve ricorrere a dei "creabili". Essi sono i quattro anelli visibili in Figura 18 ai lati (fuori dalla scena inquadrata) e le ombre di anelli al centro (come "fantasmi" nell'editor, ma invisibili una volta avviato il gioco) sono i rispettivi oggetti che chiamano la creazione dei cloni nei rispettivi punti. All'avvio i fantasmi si limitano a chiamare un clone ciascuno, in modo da ottenere gli oggetti per la scena iniziale. Per farlo, si ricorre a una variabile nascosta che rappresenta la scena corrente (Fig.19).

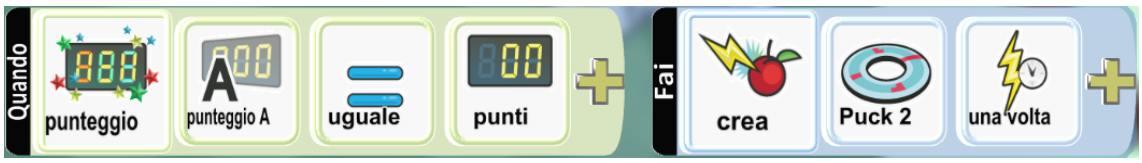


Figura 19. Esempio di utilizzo di variabile per il test.

I creabili invece contengono la logica che apparterrà ai cloni, ovvero agli oggetti effettivamente visibili a schermo e con cui si interagirà. Per esempio, quando si passa a una scena che non presentava tale oggetto (ovvero la variabile di scena assume il valore della scena inerente) si procede a eliminare quel clone.

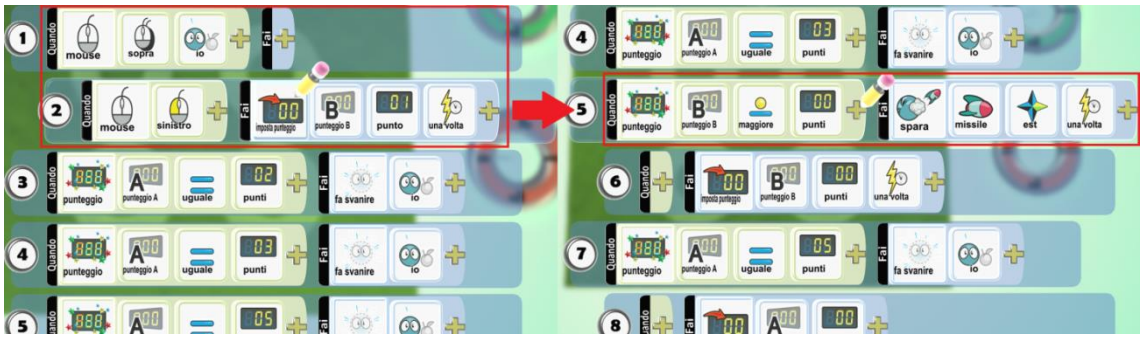


Figura 20. Oggetto che scatena azione su altro oggetto grazie a variabile.

Se era richiesto che un clic su un oggetto scatenasse un'azione su un altro, in Scratch si risolveva inviando un messaggio in broadcast e ricevendolo dall'altro oggetto. In Kodu invece si può realizzare per mezzo di una variabile nascosta. L'oggetto cliccato setta la variabile come mostrato nella Figura 20 (a sinistra) e l'altro oggetto (nella parte destra dell'immagine), quando rileva che la variabile ha assunto un certo valore, esegue le azioni richieste.

Il fantasma posizionato in alto nella Figura 18 del test è il personaggio Kodu reso invisibile. In Scratch gli script riferiti a comportamenti globali si potevano applicare allo stage, qui in Kodu non potendo farlo si ha optato per gestirli su questo personaggio fantasma, il cui script associato si occupa di ritornare alla scena iniziale quando si preme lo spazio e lo fa settando la variabile pari alla scena iniziale (Fig.21).

Figura 21. Ritorno alla scena iniziale.

Insomma, anche senza mostrare tutti i tasselli necessari alla realizzazione del test voluto, è chiaro come Kodu presenti dei limiti e complicazioni, in parte dovuti alla sua fin troppa versatilità, al punto che operazioni che all'apparenza sembrano semplici spesso non sono supportate direttamente, lasciando che l'utente le ottenga attraverso combinazioni di tasselli.

2.4 Project Spark

Project Spark è un videogioco del 2014 sviluppato dal Team Dakota e pubblicato dai Microsoft Studios per Windows 8.1, 10 e Xbox One, del genere sandbox [42]. In sostanza si tratta di un'evoluzione dei concetti introdotti in Kodu, espandendone le funzionalità, ma mantenendo la stessa identica impostazione di fondo e logica di programmazione.

Per il primo anno si basava sul supporto di micro-transazioni e successivamente ha reso totalmente gratuiti i suoi contenuti. Nel 2016 Microsoft ha interrotto lo sviluppo di Project Spark, non rendendo più possibile scaricare il gioco e chiudendo i server dei servizi online, così per gli utenti si è persa la principale funzionalità che era quella di poter condividere online i propri progetti. Ora i giocatori su Xbox che hanno il gioco possono continuare a giocare localmente, mentre gli utenti PC possono scambiarsi i progetti grazie a un'applicazione esterna.

Project Spark mette a disposizione dell'utente una tavolozza di strumenti per creare il proprio gioco personalizzato in un ambiente tridimensionale, gestendo le varie meccaniche, e in passato permetteva di condividere i risultati ottenuti con altri utenti, con la possibilità di scaricare le creazioni di altre persone per poterle modificare a propria volta. L'utente ha a disposizione una grande varietà di personaggi e oggetti per il suo gioco, che condividono lo stesso stile grafico fantasy. Oltre a creare l'ambiente di gioco e gli oggetti al suo interno, Project Spark permette di registrare audio e anche definire le animazioni attraverso il sensore Microsoft Kinect [50].

Ci sono tuttavia dei fattori che lo distinguono da Kodu. Kodu è più focalizzato sull'educazione, mentre Spark in quanto videogioco puro si rivolge più sull'intrattenimento, con un aspetto grafico più accattivante per rivolgersi a un target di utenti di età leggermente superiore. Infatti, se Kodu era adatto ai bambini, le funzionalità aggiuntive di Spark lo rendono più complesso e uno strumento di creazione giochi più flessibile. Inoltre si ricorda che Kodu è attualmente attivo per i sistemi Windows e continua a ricevere aggiornamenti, mentre Spark era disponibile per Windows 8 e 10 e Xbox One, ma non si può più scaricare.

Per quel che concerne la programmazione, entrambi utilizzano lo stile "When-Do", ma Spark possiede un numero notevolmente maggiore di tasselli, che si traduce in un linguaggio migliore per esempio nel realizzare operazioni matematiche (utili per gestire i vettori di spostamento degli oggetti), un ambito nel quale la controparte di Kodu si rivela più debole.

2.4.1 Interfaccia utente

Il menu principale, attraverso il quale avviare il gioco attuale, modellare il terreno o creare gli oggetti, in Kodu era visualizzato nella parte inferiore dello schermo, mentre in Spark è sulla sinistra, ma a grandi linee svolge le stesse funzioni principali. Nella parte bassa sono visualizzabili gli oggetti più utilizzati per averli più a portata e non doverli cercare ogni volta dalla grande lista di quelli esistenti (Fig.22).

Figura 22. Interfaccia di Spark.

Per il presente lavoro di tesi non è particolarmente interessante analizzare la parte grafica di creazione dell'ambiente di gioco, che si presenta come una versione semplificata di altre controparti in prodotti professionali quali l'editor di Unity, il noto motore grafico per lo sviluppo di videogiochi. Invece la logica di gioco in Project Spark è definibile per mezzo di un'interfaccia pulita ma al contempo ricca di funzionalità e versatile da permettere di ottenere risultati anche complessi e strutturati.
Il linguaggio usato in Project Spark per aiutare i giocatori a programmare come gli oggetti interagiscano è la naturale evoluzione delle stesse funzionalità presenti in Kodu: l'impostazione è la stessa vista in precedenza per Kodu e programmare in Spark richiede di usare l'identico approccio "If This, Then That", dicendo agli oggetti quando e come reagire alle differenti situazioni, le quali possono essere anche piuttosto stratificate (Fig.23). I tasselli a disposizione sono sempre suddivisi in categorie e sottocategorie con selettori circolari e con una finestra di descrizione testuale a fianco che ne spiega meglio la funzione, ma qui sono in numero molto maggiore, dell'ordine delle centinaia. Per trovare quello che si desidera utilizzare può venire in aiuto un campo di ricerca per applicare un filtro sul loro nome.

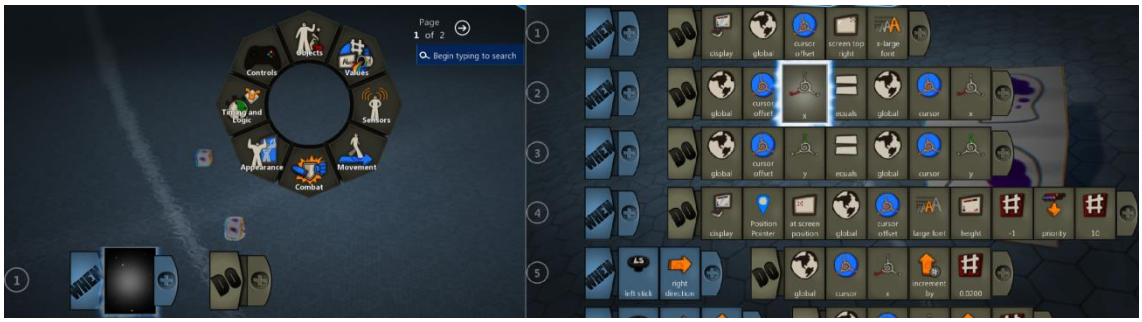


Figura 23. Esempi di selezione di un When e di righe di script generate.

Come in Kodu, Spark permette di copiare una riga per duplicarla più facilmente o spostarla verso destra per annidarla a quella precedente. Anche qui le regole sono distribuibili su più pagine, ma in più è possibile dare un nome alle pagine per facilitare la distinzione delle varie "scene" di gioco desiderate. Tra le funzionalità principali di Project Spark ci sono i "Brain" (cervelli). Si tratta essenzialmente di una serie di righe di codice racchiuse in un solo tassello e che generano obiettivi, funzioni e proprietà a tutti gli elementi in gioco. Assegnando un brain, il giocatore è in grado per esempio di modificare il comportamento di un oggetto come una roccia dicendo che segua il giocatore quando lo vede. Esiste un gran numero di brain già esistenti messi a disposizione per varie necessità. Un personaggio Goblin di default ha un comportamento aggressivo che attacca il giocatore, ma è ovviamente possibile intervenire sul suo comportamento assegnandogli ad esempio il brain di un uccello per farlo volare via. Si può in sostanza far corrispondere i brain alle classiche funzioni della programmazione classica: blocchi di codice richiamabili in più circostanze per evitare di doverli riscrivere ogni volta. Un altro strumento utile è il vettore, che permette di definire delle coordinate specifiche dello spazio tridimensionale, dove per esempio indicare di spostare un oggetto.

2.4.2 Test

Le differenze nei tasselli messi a disposizione rispetto a Kodu hanno portato a ricreare il test desiderato attraverso una impostazione generale della logica del codice differente.
Come oggetti si è ricorso a un cubo, una sfera, un anello e una testa di scimmia che fa le veci dell'oggetto Suzanne di Blender, posizionati in aria e inquadrati frontalmente, mentre fuori dalla vista della camera sono aggiunti due ulteriori cubi che conterranno il codice di logiche più generiche. Infatti, in Spark si ricorre spesso a cubi logici non visibili il cui solo scopo è fare da contenitori per codice specifico, soprattutto in quei giochi a schermata fissa.

Un primo cubo si occupa di gestire i cambi scena: per ogni sua pagina si associa una scena e si dichiara che “when” si carica la scena, “do” elimina gli oggetti della scena precedente per generare quelli della scena nuova. Il secondo cubo si occupa invece dei comportamenti globali indipendenti dalla scena attuale, in questo caso fa sì che in ogni istante, alla pressione di un tasto specifico, si ricarichi la scena iniziale (Fig.24). Infine un grande cubo blu fa da sfondo e contiene il codice per la camera fissa.

Figura 24. Il codice di ricarica della scena iniziale e una pagina di gestione cambio scena.

Figura 25. Script per gli oggetti del test.

Come sintetizzato nella Figura 25, per il cubo e per Suzanne un clic su di essi scatena il cambio scena nel cubo logico addetto alla gestione delle scene e un clic sulla sfera esegue un effetto sonoro. Invece un clic su torus scatena un’animazione su Suzanne per mezzo di una variabile booleana, su cui Suzanne sta in ascolto per avviare la sua animazione di rotazione.

L’oggetto designato quale puntatore non è rilevante descriverlo nel dettaglio: per realizzarlo si può ricorrere a un’icona 2D ed esisterebbe un tassello di raycasting, ma è noto dare problemi di rilevazione e non funziona correttamente. Una possibilità per spostarlo sullo schermo è quella di scrivere numerose regole che muovano un vettore cursore gradualmente, per ognuna delle possibili direzioni, e imporre al puntatore di avere le coordinate di questo vettore. Un clic sarà riconosciuto su un particolare oggetto se si rileva che il puntatore si trovava sopra di esso.

Capitolo 3

Tecnologie

In questo capitolo si analizzano le tecnologie interessate dal progetto di tesi. Si inizia con un'ampia analisi del programma di modellazione Blender, soffermandosi in particolare sul suo Game Engine. Successivamente si presenta la teca olografica dove saranno visualizzate le scene finali e il dispositivo Leap Motion impiegato per l'interazione gestuale, concludendo con la descrizione del linguaggio di programmazione JavaFX, che è stato scelto per realizzare il presente programma.

3.1 Blender

Blender è una suite per la creazione di contenuti 3D open source e gratuita. Può essere usato per creare visualizzazioni 3D come immagini fisse, video e videogiochi interattivi real-time.
Blender ha un'ampia varietà di strumenti che lo rendono adatto a quasi qualsiasi tipo di produzione multimediale. Si adatta bene a singoli utenti e piccoli studi che beneficiano della sua pipeline unificata e del processo di sviluppo responsive. Le persone e gli studi in giro per il mondo lo usano per i loro progetti personali, per video commerciali, per film, giochi, ricerca scientifica e altre applicazioni interattive.
Offre una vasta gamma di strumenti essenziali, inclusi modellazione, rendering, animazione, illuminazione, video editing, VFX, compositing, texturing (mappatura UV delle texture), rigging (scheletratura dei modelli), e vari tipi di simulazioni, come la simulazione di corpo rigido, di fluidi e sistemi particellari [51]. Inoltre fornisce una piattaforma integrata per la creazione di giochi. Le funzionalità che presenta lo rendono un valido rappresentante, per caratteristiche e profondità, della categoria di programmi dedicati alla modellazione 3D come Cinema 4D, Maya, 3ds Max, LightWave, Modo.
Blender è un'applicazione multipiattaforma, per sistemi Windows (Vista e superiori), macOS (OSX 10.6 e superiori) e Linux ed esiste come porting per FreeBSD. Rispetto alle altre suite di creazione 3D ha anche dei requisiti relativamente bassi di memoria e sistema. La sua interfaccia usa OpenGL per fornire un'esperienza uniforme su tutte le piattaforme e gli hardware supportati ed è customizzabile con script Python.

In origine, Blender venne sviluppato nel 1994 come applicazione interna da parte dello studio di animazione olandese NeoGeo. NeoGeo era stato fondato nel 1988 da Ton Roosendaal [52] assieme ad altri soci e rappresentava il più grande studio di animazione in Olanda e uno dei principali in Europa. Nel giugno del 1998 Roosendaal fondò la società Not a Number Technologies (NaN) come spin-off di NeoGeo per continuare lo sviluppo di Blender, al fine di renderlo uno strumento fruibile anche da artisti esterni all'azienda, e distribuirlo inizialmente come software proprietario gratuito. Per l'epoca, l'idea di rendere disponibile in modo totalmente gratuito un software di modellazione del genere rappresentava un'idea coraggiosa, in quanto la maggior parte dei modellatori in commercio costava diverse migliaia di dollari. Il successo fu istantaneo, ma nel 2002 la società NaN finì in bancarotta e i creditori consentirono la pubblicazione di Blender quale software libero, secondo i termini della licenza GNU General Public License. Poco prima di chiudere NaN, rilasciarono un'ultima release di Blender in cui inclusero, come una sorta di easter egg, un modello 3D semplificato di una testa di scimpanzé, di nome Suzanne. Sempre nel 2002 Roosendaal fonda successivamente l'organizzazione no-profit Blender Foundation e completa con successo una campagna per la raccolta fondi, grazie alla quale pubblica il codice sorgente di Blender. Così ora Blender è un progetto open source molto attivo ed è guidato dalla Blender Foundation. Anche se venne inserito come uno scherzo, Suzanne è ancora oggi presente in Blender e spesso utilizzato come modello di test delle varie funzioni.

Nel 2008 Blender ha iniziato un ampio processo di riscrittura per migliorare le sue funzioni e il flusso di lavoro, culminato nel 2011 quando si ha assistito a un significativo aggiornamento dell'interfaccia utente. Il suo sviluppo non ha subito pause lungo questi anni, per merito di un nutrito gruppo di volontari in tutto il mondo. Attualmente è arrivato alla versione 2.79 del 12 settembre 2017 e continua a ricevere regolari aggiornamenti.

Blender è stato utilizzato per la realizzazione di pubblicità televisive in tutto il mondo ed è usato anche dalla NASA per suoi modelli 3D pubblici o applicazioni. Nel mondo cinematografico, il primo grande progetto che ne fece uso fu Spider-Man 2, per creare gli animatic e lo storyboard, e successivamente è stato impiegato in vari film e serie TV per gli effetti speciali e in film d'animazione. Inoltre nel corso degli anni la Blender Foundation ha sviluppato numeri progetti, utili anche per pubblicizzare Blender esaltandone le capacità: il corto animato "Elephants Dream", seguito dal corto "Big Buck Bunny", il videogioco "Yo Frankie!" (Fig.26), i film in computer grafica "Sintel" e "Tears of Steel", mentre è attualmente in sviluppo da diversi anni il film "Cosmos Laundromat" di cui per ora ne è stato realizzato un corto.

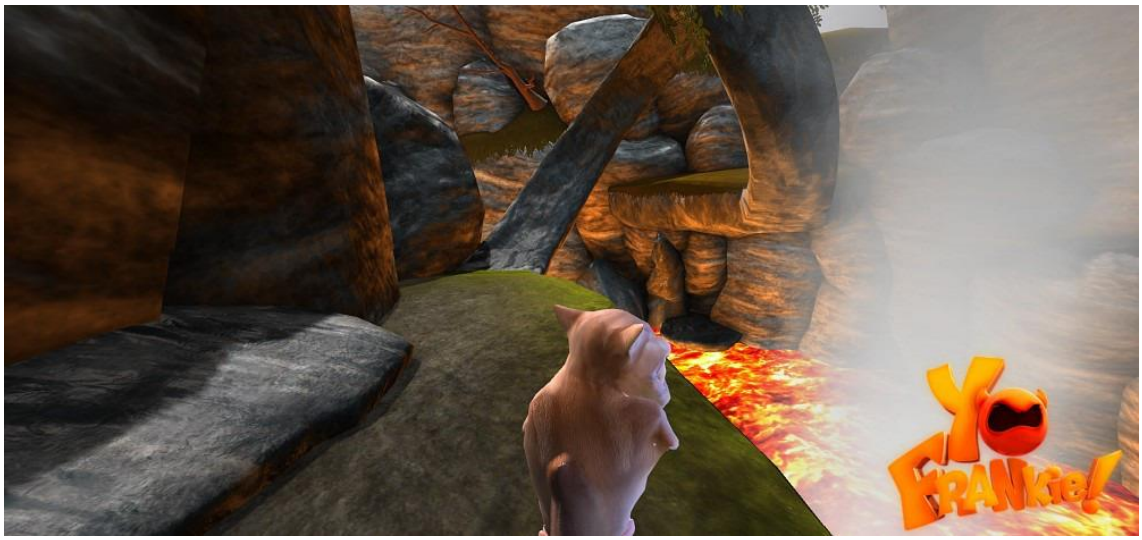


Figura 26. Screenshot da “Yo Frankie!”, realizzato con il Blender Game Engine. (fonte: wiki.blender.org)

3.1.1 Funzionalità

Un progetto di Blender viene salvato in un formato di file “.blend”, che include in un singolo file tutte le scene definite dall’utente. Tutti i file “.blend” sono compatibili tra le varie piattaforme e tra le differenti versioni di Blender, con delle eccezioni: dalla versione 2.5 le animazioni sono memorizzate nel file in maniera non compatibile con versioni precedenti di Blender, mentre le mesh a partire dalla versione 2.63 hanno un nuovo formato non comprensibile dalle precedenti versioni.
In un “.blend” sono salvate le scene, gli oggetti, i materiali, le texture, i suoni, le immagini, le animazioni e tutti gli altri effetti presenti. Eventuali dati importati dall’esterno come immagini o suoni sono anche memorizzati facendo riferimento al loro path assoluto o relativo. Inoltre un “.blend” possiede anche le configurazioni dell’interfaccia che è stata impostata dall’utente.
Blender organizza i dati in varie tipologie di “data block”, come Object, Mesh, Lamp, Scene, Material, Image e così via [53]. Un oggetto in Blender consiste in più data block, ovvero un oggetto rappresentato da una mesh poligonale comprenderà almeno un Object e una Mesh e solitamente anche un Material e molti altri, connessi tra loro. Questo permette di collegare vari data block tra loro, per esempio si può avere più oggetti che condividono la stessa Mesh, in modo che modificando la mesh condivisa si vada a modificare tutti questi Object. Oggetti, mesh, materiali, texture, ecc. si possono anche importare da altri file “.blend”, file che dunque sono usabili anche come librerie di risorse riutilizzabili.
Tra le funzioni di Blender si ricordano le seguenti.

- Disponibilità di numerose primitive geometriche comprese le mesh poligonali (dai cubi alle icosfere) e includendo una veloce gestione della suddivisione delle superfici, le curve di Bezier e NURBS (Non Uniform Rational B-Splines) utili anche per determinare un percorso, le superfici NURBS, le metaball e i font vettoriali.
- Modificatori per applicare effetti interattivi e non distruttivi, in quanto sono operazioni automatiche che non coinvolgono la geometria base dell'oggetto interessato, ma solo il suo aspetto. Permettono all'utente di evitare tediose operazioni manuali e si possono combinare ed eseguire nell'ordine desiderato.
- Possibilità di convertire da e verso diversi altri formati relativi ad applicazioni 3D, permettendo di importare o esportare un file per esempio in Collada (.dae), 3D Studio (.3ds), FBX (.fbx) o Wavefront (.obj).
- Gestione delle animazioni per spostare o deformare gli oggetti, per esempio con la deformazione lattice, con la possibilità di creare armature con scheletri, applicare keyframe, vincoli alle proprietà degli oggetti, morph target animation, la cinematica inversa, le animazioni non lineari, la definizione di percorsi, il calcolo pesato dei vertici e una simulazione fisica che include la possibilità da parte delle mesh di gestire le particelle (sistema particellare usabile ad esempio per fare i capelli).
- Strumenti di gestione della fisica, come la simulazione per la dinamica del corpo rigido e morbido, inclusa rilevazione delle collisioni tra mesh, dinamica dei fluidi (attraverso i metodi reticolari di Boltzmann), simulazione del fumo, della pioggia, della polvere, dei capelli o erba e la simulazione dei vestiti.
- Controllo real-time durante la simulazione fisica e il rendering.
- Gestione dell'editing audio e video non lineare. Blender presenta un editor video chiamato Video Sequence Editor (VSE), che ha varie caratteristiche come effetti quali Gaussian Blur, color grading, transizioni Fade e Wipe e altre trasformazioni video. Tuttavia, con VSE non c'è supporto multi-core per il rendering video.
- Due differenti motori di rendering interni, tra cui Cycles, introdotto a partire da Blender 2.61 e che dalla versione 2.65 supporta l'Open Shading Language (OSL) con cui l'utente può creare i propri nodi. Cycles a differenza dell'altro motore impostato di default, è di tipo unbiased e si avvantaggia della GPU per velocizzare i tempi di rendering. Ci sono due modalità di rendering con la GPU: CUDA, preferito per le schede grafiche NVIDIA, e OpenCL, che supporta il rendering sulle schede grafiche AMD. Cycles viene fornito installato come un add-on, per essere utilizzato lo si deve settare come motore di rendering attivo.

- Il motore di rendering real-time Blender Game Engine permette di gestire le interazioni con gli oggetti attraverso “blocchi logici”, consentendo di realizzare videogiochi, simulazioni, programmi indipendenti (stand-alone) o altri contenuti in tempo reale.
- Possibilità di estendere le funzionalità disponibili attraverso script nel linguaggio Python, utili a esaminare o automatizzare le differenti caratteristiche, come gestire la logica di gioco.
- Texture procedurali e basate su nodi, come anche vertex painting, weight painting, texture painting, dynamic painting e projective painting.
- Camera tracking e object tracking.
- Blend4Web, un framework WebGL open source che può essere usato per convertire le intere scene di Blender, con grafica, animazioni, suoni e fisica, per funzionare nei browser web standard.

3.1.2 Interfaccia

Blender si basa su uno spazio di lavoro interamente a oggetti. La sua interfaccia è composta da un certo numero di aree ridimensionabili, contenenti una desiderata tipologia di editor di Blender, come la vista 3D o le proprietà di un oggetto selezionato. Ciascuna area è suddivisibile a sua volta in altre due, per aumentare gli editor contemporaneamente a schermo (o viceversa fondibili in una sola) e son sempre tutte visibili in quanto non sono sovrapponibili. La composizione di tali aree dell’interfaccia è personalizzabile dall’utente, permettendo quindi di disporre al meglio il proprio spazio di lavoro, in base al compito specifico che si intende svolgere, e modificando l’interfaccia quando si ritiene opportuno, per esporre soltanto le caratteristiche d’interesse. Se ad esempio ci si deve occupare solo di video editing o UV mapping o texturing, si possono nascondere tutte le altre caratteristiche non richieste sul momento.
Avviando Blender, si ha una finestra simile a quella visualizzata nella Figura 27. L’interfaccia utente è consistente tra le varie piattaforme.

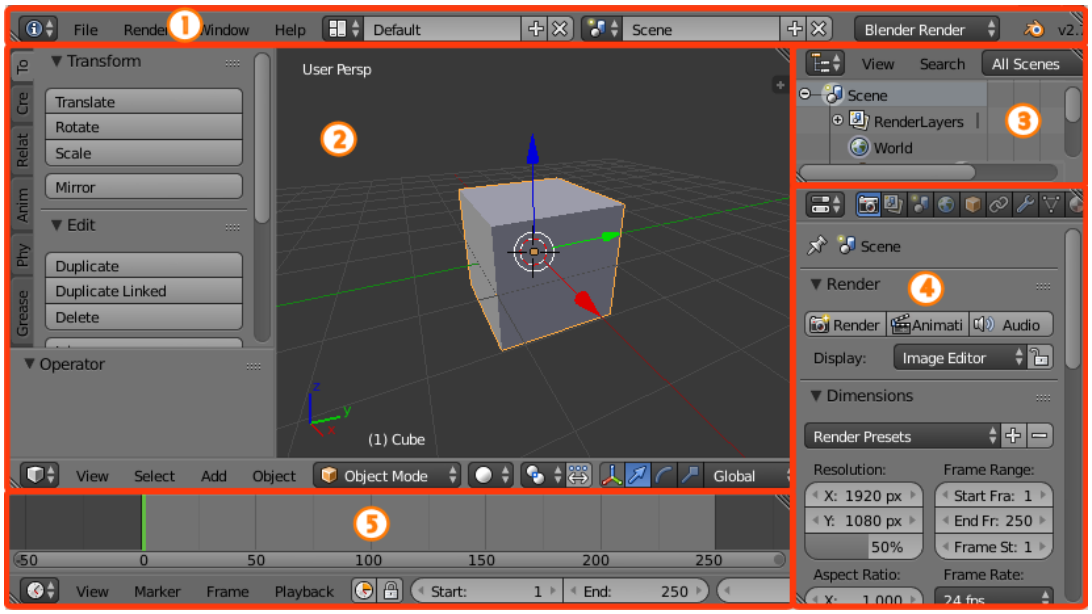


Figura 27. Schermata iniziale di Blender con i cinque editor di partenza: Info Editor (1), 3D View (2), Outliner (3), Properties Editor (4), Timeline Editor (5).

Si può personalizzare la schermata scegliendo quali editor visualizzare tra quelli indicati di seguito.

Settings

- Info Editor: da dove per esempio poter gestire il file, selezionare il layout degli editor o il motore di rendering.
- Properties Editor: dove gestire tutte le proprietà della scena e dell'oggetto attualmente selezionati suddivise in vari tab, come per esempio il materiale assegnato all'oggetto, le texture, i modificatori o la fisica da applicare.
- Outliner: mostra in una struttura ad albero tutte le scene e gli oggetti presenti nel file.
- User Preferences: contiene le impostazioni per controllare come si comporta Blender, per esempio decidendo l'aspetto degli elementi dell'interfaccia grafica, scegliere come mappare i tasti per le scorciatoie dei comandi, personalizzare l'aspetto e i colori dell'interfaccia, impostare le preferenze di risoluzione, suono e così via.

3D

- 3D View: che mostra la scena con gli elementi grafici al suo interno.

Animation

- Timeline Editor: per scorrere i frame e selezionare gli intervalli temporali in cui avviare le animazioni.

- Graph Editor: per modificare i driver e l'interpolazione tra i keyframe.
- Dope Sheet: per modificare le tempistiche dei keyframe.
- NLA Editor: l'editor per Non-Linear Animation permette di modificare con facilità le animazioni in modo generico, senza intervenire direttamente sui singoli keyframe.

Image/Video

- UV/Image Editor: per vedere e modificare le immagini e le mappe UV delle texture.
- Video Sequence Editor: dove si hanno gli strumenti per l'editing dei video.
- Movie Clip Editor: con gli strumenti per il motion tracking.

Nodes/Logic

- Text Editor: permette principalmente di scrivere gli script Python.
- Node Editor: editor con un flusso di lavoro basato su nodi.
- Logic Editor: permette di decidere la logica di gioco per i vari oggetti presenti.

Other

- File Browser: usato nelle varie operazioni sui file, come salvare o aprire un ".blend".
- Python Console: console per eseguire velocemente i comandi, con accesso all'API Python.

Nella 3D View, le due modalità principali di editing degli oggetti in Blender sono la Object Mode (modalità oggetto) e la Edit Mode (modalità modifica). Si può passare dall'una all'altra premendo il tasto Tab. In Object Mode è possibile manipolare un oggetto intero, per esempio per spostarlo, scalarlo e ruotarlo nell'insieme, mentre in Edit Mode si può intervenire sui suoi singoli vertici della mesh. Oltre a queste due modalità ne esistono molte altre, come il Vertex Paint (pittura vertici), Weight Paint (pittura pesi) e Sculpt Mode (modalità scultura).
L'interfaccia utente di Blender si basa sui principi di:

- Non sovrapposizione: l'interfaccia utente è progettata per permettere di vedere tutte le opzioni e strumenti rilevanti all'istante, senza dover spostare o trascinare in giro gli editor.
- Non bloccante: gli strumenti e le opzioni dell'interfaccia non bloccano l'utente in nessuna sezione di Blender. Blender tende a non usare finestre pop-up per richiedere all'utente di inserire dei dati prima di eseguire un'operazione.

- Non modale: gli strumenti sono facilmente accessibili senza richiedere tempo per selezionarli da una lista di differenti strumenti e l'input dell'utente dovrebbe rimanere il più coerente e prevedibile possibile, senza dover cambiare i metodi comunemente utilizzati (mouse e tastiera).

Blender ha sempre avuto la fama di essere un programma difficile da imparare. Questo perché quasi tutte le funzioni che presenta possono essere richiamate con scorciatoie di combinazioni di tasti da tastiera e dunque quasi ogni tasto è collegato a una particolare funzione. In passato Blender risultava poco immediato per via del fatto che i comandi richiedevano la conoscenza delle numerose scorciatoie da tastiera, infatti fino alla versione 2.3x non vi era un altro sistema con cui poter interagire e questo spiega come mai Blender sia rimasto a lungo noto come un programma poco immediato e ostico da apprendere.
Invece oggi si ha anche a disposizione menu più perfezionati e comprensibili, che consentono di svolgere praticamente tutte le operazioni anche attraverso il mouse. Le varie versioni recenti hanno migliorato l'interfaccia, rendendola più intuitiva per chi è alle prime armi col programma. Negli ultimi anni la GUI è stata notevolmente modificata, permettendo ora di cambiare il colore, di usare widget trasparenti, una più evoluta gestione dell'albero degli oggetti e altre piccole migliorie.
Anche se non dispone di un tutorial iniziale, Blender è documentato in modo esaustivo sul suo sito web, oltre a disporre di discussioni su forum e molti video tutorial su YouTube.

3.1.3 Blender Game Engine

Il Blender Game Engine (BGE) è un componente di Blender, utilizzato per realizzare contenuti interattivi real-time, da visualizzazioni architettoniche a simulazioni di giochi. Il motore di gioco è stato scritto da zero in C++ come componente sostanzialmente indipendente e include il supporto per caratteristiche come lo scripting in Python e suoni 3D OpenAL. Il Game Engine può simulare il contenuto attraverso Blender, ma include anche l'abilità di esportare un binario di runtime per Linux, macOS e MS-Windows.
Il BGE venne sviluppato nel 2000 da Erwin Coumans e Gino van der Bergen. Il loro scopo era quello di realizzare un prodotto intuitivo e semplice per la creazione dei giochi e altri contenuti interattivi. I giochi prodotti sono eseguibili sia come applicazioni stand-alone o integrati in una pagina web per mezzo di un plugin apposito. Come Blender, il BGE usa OpenGL, una specifica dichiarante una API per più piattaforme e linguaggi, rivolta alla scrittura di applicazioni per la produzione di computer grafica 2D e 3D. Attualmente si discute di grossi cambiamenti che verranno apportati al BGE in un prossimo

futuro, nella versione di Blender 2.8, principalmente rivolti a fondere maggiormente il BGE con Blender, senza più dover chiamare il game engine, ma passando a una sorta di "Interaction mode".
La principale differenza tra il Game Engine e il sistema Blender convenzionale è nel processo di rendering. Nel normale motore di Blender, le immagini e le animazioni sono generate off-line, una volta renderizzate non possono più essere modificate. Al contrario, il BGE renderizza le scene continuamente in real-time e incorpora strutture per l'interazione utente durante il processo di rendering. Il motore di gioco si avvantaggia di varie librerie, come Audaspace per l'audio, Bullet per la fisica, Detour per il path-finding e spatial reasoning e Recast per la costruzione di mesh di navigazione per i giochi.
Il BGE sovrintende un ciclo di gioco, che processa la logica, i suoni, le simulazioni fisiche e di rendering in ordine sequenziale. Esso è progettato per astrarre le complesse caratteristiche del motore in una semplice interfaccia utente, che non richiede competenze di programmazione.
Nel realizzare un gioco o simulazione, si inizia creando gli elementi visuali renderizzabili, siano essi modelli 3D o immagini, poi si procede definendo le interazioni nella scena determinano i comportamenti personalizzati e come questi vengano invocati. Successivamente si crea almeno una camera per avere un frustum di vista da cui poter renderizzare la scena e si modificano i parametri per la resa dell'ambiente, come per esempio il rendering stereo. Infine si po' avviare il gioco, attraverso il player interno o esportando un runtime.
L'utente grazie al BGE ha accesso a un potente Logic Editor event driven di alto livello, che usa un sistema di componenti specializzati detti "Logic Bricks" ("mattoni logici"), per controllare il movimento e la visualizzazione degli oggetti di gioco. Questo metodo di definizione della logica di interazione per mezzo di blocchi si rivela semplice e intuitivo anche per chi non è abituato a programmare, rendendo leggibili le varie relazioni causa-effetto e i comportamenti di azione-reazione presenti negli oggetti dell'ambiente di gioco. Pertanto si definisce come un input dell'utente via mouse o tastiera possa scatenare eventi di gioco, che a loro volta ne possono scatenare altri. Per esempio, si può creare un gioco in cui cliccando col mouse sui nemici si incrementi un punteggio, definire un comportamento ai nemici oppure fare in modo che quando un particolare nemico viene sconfitto si passi alla scena di gioco successiva.
I mattoni logici grafici si distinguono in "sensori", "controllori" e "attuatori", definiti su colonne distinte e collegati tra loro in sequenza. L'utente prima li genera e dopo procede a connettere i sensori ai controllori e i controllori agli attuatori tirando le linee opportune. I sensori rappresentano gli eventi scatenanti una qualche azione, agendo da trigger e inviando un impulso positivo a tutti i controllori collegati, i quali si occupano di collegare i dati inviati dai sensori, specificando lo stato in cui possono operare. Eseguite le operazioni logiche indicate, i controllori inviano un segnale di impulso agli attuatori a cui sono connessi. Ovvero quando un sensore è attivato manda un impulso positivo e quando è disattivato ne manda uno negativo, mentre il controllore controlla e combina questi impulsi multipli in ingresso e, in base alla logica booleana impostata in esso, genera il relativo risultato. Infine gli attuatori

si occupano di eseguire azioni, quali ad esempio muovere un oggetto, crearne un altro o generare un suono, attivandosi quando ricevono un impulso positivo da uno dei controllori collegati a monte. Inoltre il motore di gioco può anche essere esteso, espandendo le funzionalità del logic editor, attraverso script Python, come nel caso in cui si necessiti di controllori più complessi della logica booleana messa a disposizione.
Assieme ai mattoni logici, si hanno a disposizione anche delle "properties", che sono l'equivalente delle classiche variabili nei linguaggi di programmazione. Un oggetto può avere associato una o più property in cui salvare dati relativi ad esso, come il numero di munizioni, la vita, il nome e così via.
Infine si segnala la possibilità di definire diversi stati a un oggetto. Durante l'esecuzione di un gioco, lo stato corrente di un oggetto ne definisce il comportamento: un personaggio può ad esempio trovarsi in uno stato attivo, di riposo o inattivo e passare da uno all'altro in qualsiasi istante, il suo comportamento dipenderà dallo stato attuale che gli fa eseguire i mattoni logici desiderati. Gli stati sono definibili attraverso la sola sezione dei controllori (e non sensori o attuatori) e ogni oggetto dispone di fino a 30 stati possibili, ma può trovarsi in un solo stato alla volta. Nei giochi più semplici non serve intervenire sugli stati, altrimenti si deve ricorrere a un attuatore di tipo "stato" che durante il gioco scatena il passaggio a uno stato differente. Se si ha definito più di uno stato, si può scegliere quale debba essere quello iniziale all'avvio del gioco.
Si procede ora ad analizzare nel dettaglio i sensori, controllori e attuatori esistenti, che sono le fondamenta su cui si basano le interazioni realizzabili nel progetto di tesi. Infatti l'applicazione Visual Scene Editor realizzata si prefigge lo scopo di nascondere agli utilizzatori finali il funzionamento di queste tre tipologie di mattoni logici del Logic Editor di Blender.

3.1.3.1 Sensori

La prima colonna di mattoni logici è composta dai sensori, che come detto rappresentano i trigger degli eventi desiderati. Essi generano un segnale in uscita quando rilevano un qualche evento scatenante, come ad esempio la pressione di un tasto della tastiera, il cambiamento di valore in una proprietà o la collisione di un oggetto con un altro. Se un sensore viene attivato, si invia un impulso positivo a tutti i controllori che sono connessi ad esso in cascata.
Per prima cosa, in alto è presente una testata della colonna con vari controlli che fanno da filtro sui sensori da visualizzare, utile per nascondere quelli non necessari e tenere visibili solo quelli che interessano. Subito sotto è presente la testata dell'oggetto con indicato il nome dell'oggetto attualmente selezionato e a fianco ad esso c'è un menu a tendina che permette di scegliere il tipo di sensore da

aggiungere. Se più di un oggetto è correntemente selezionato, questa riga sarà ripetuta per ciascuno, in modo da raggruppare i mattoni logici per i vari oggetti.
Ogni sensore generato presenta una serie di pulsanti, campi e menu comuni a tutti, evidenziati nella Figura 28.

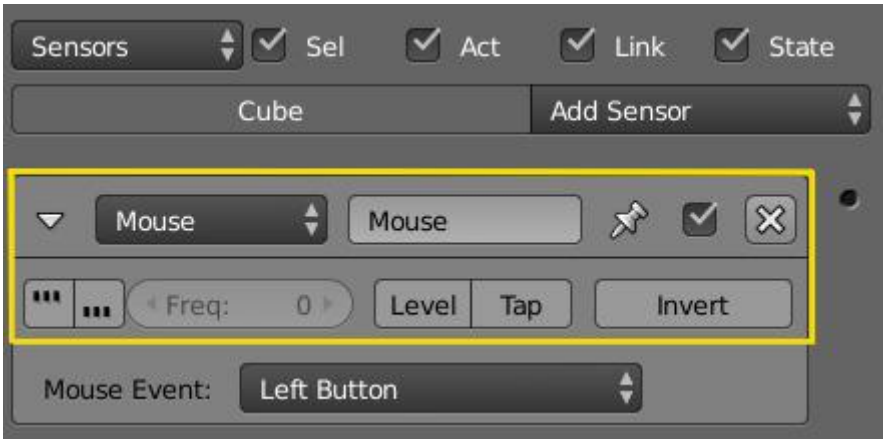


Figura 28. Opzioni comuni di un sensore.

Si ha una prima riga composta da un pulsante a triangolo che collassa tutte le informazioni del sensore in una singola linea, seguito da un menu indicante il tipo di sensore presente e dal nome personalizzabile associato a questo sensore; seguono un pulsante di pin per dire se mostrare il sensore anche se non è collegato a un controllore di stato visibile, una checkbox per disattivare il sensore e un pulsante per eliminarlo. Nella seconda riga si specifica come deve comportarsi il sensore nel triggerare i controllori ad esso collegati. Questo avviene quando il sensore cambia il suo stato da negativo a positivo o viceversa, ma anche quando cambia da disattivo a attivo. Se si interviene sui primi due pulsanti, quelli con i tre puntini alti o bassi, si può impostare che il sensore triggeri i controllori connessi per tutto il tempo in cui il suo stato è positivo o negativo (il base al pulsante selezionato) a intervalli di tempo pari al numero di frame settati nel campo “Freq”; i pulsanti “Level” e “Tap” servono rispettivamente per dire se triggerare i controllori quando si ha un cambiamento di stato o solo per un istante, anche se il sensore resta “vero”. Infine un pulsante “Invert” permette di invertire l’output del sensore.
Esistono varie tipologie di sensori che è possibile definire, come riportato di seguito.

- Actuator: rileva quando un particolare attuatore riceve un impulso di attivazione. Quando l’attuatore specificato diventa attivato o disattivato, viene mandato un impulso rispettivamente vero o falso.
- Always: come il nome lascia intuire, questo sensore risulta sempre attivo e manda un output di segnale continuo a intervalli di tempo regolari. Utile per quelle azioni che vanno eseguite con

una frequenza costante oppure, se si seleziona il pulsante “Tap” descritto in precedenza, questo Always permette di scatenare un’azione all’avvio del gioco.

- Collision: riconosce le collisioni che avvengono tra oggetti o materiali. Si può specificare di rilevare solo le collisioni con altri oggetti che possiedono una certa property o materiale specificato, altrimenti se non si dichiara nulla rileva le collisioni con qualsiasi oggetto, generando un impulso positivo. Un pulsante “Pulse” permette di rendere l’oggetto sensibile ad altre collisioni anche se è ancora in contatto con l’oggetto che ha generato l’ultimo impulso positivo. Purtroppo questo sensore non può riconoscere le collisioni con i soft body, una limitazione data dalla libreria Bullets usata dal Game Engine.
- Delay: ritarda l’output di uno specifico numero di cicli logici. Può essere utile quando serve che un’altra azione venga eseguita per prima o per temporizzare gli eventi. Permette di indicare il numero di cicli logici di attesa prima di inviare l’impulso positivo e la durata dopo la quale mandare l’impulso negativo finale. Inoltre è possibile indicare di renderlo ciclico.
- Joystick: riconosce i movimenti di un joystick specifico. Permette di indicare quale joystick usare e filtrare su un certo evento del joystick piuttosto che tutti quelli di un certo tipo.
- Keyboard: rileva gli input da tastiera. Invia un impulso positivo quando riconosce la pressione del tasto su cui è in ascolto, con la possibilità anche di definire combinazioni di tasti (per esempio CTRL + ALT + T), mentre un impulso negativo quando si rilascia il tasto. Permette anche di salvare l’input immesso da tastiera in una property di tipo String. Inoltre si può utilizzare questo sensore per assegnare un valore ad una property del Game Engine, dichiarando il nome della property nel campo “Target” e l’input sarà attivo finché la property booleana indicata nel campo “LogToggle” sarà impostata a true.
- Message: rileva messaggi di testo in ricezione o valori assunti dalle proprietà. Il sensore manda un impulso positivo quando riceve un messaggio, inviato da un qualsiasi oggetto mediante l’attuatore Message, con la possibilità di filtrare sui messaggi che presentano uno specifico campo “Subject”.
- Mouse: riconosce gli eventi del mouse, come movimenti, clic e passaggi del mouse sopra un oggetto.
- Near: rileva la presenza di oggetti in movimento entro una particolare distanza da se stessi. Nel campo “Distance” si specifica la distanza entro cui si rilevano gli oggetti vicini, inviando un impulso positivo, e nel campo “Reset” la distanza oltre la quale un oggetto vicino non viene più identificato. Come per il sensore Collision, si può filtrare gli oggetti in base alle property e anche in questo sensore i soft body non sono rilevabili.
- Property: rileva modifiche nei valori delle proprietà presenti nell’oggetto corrente. Permette di testare il valore presente in una property, come per esempio valutare se è uguale o maggiore a un

certo valore immesso o semplicemente rilevarne un cambiamento. Si può anche confrontare una property con un'altra.

- Radar: individua gli oggetti presenti all'interno di un certo raggio di distanza dall'oggetto attuale, ma si differenzia da Near perché controlla solo entro un angolo di un asse, formando un cono di vista immaginario. Riesce a rilevare oggetti anche se coperti da altri, ma un oggetto per essere rilevabile deve essere settato come "Actor". Anche qui è possibile filtrare gli oggetti in base alla presenza o meno di una property.
- Random: Genera impulsi casuali, con un campo in cui poter indicare un seme random da applicare.
- Ray: emette un raggio nella direzione di un asse e rileva collisioni con esso. Può essere filtrato per individuare soltanto gli oggetti con un certo materiale o property. Attivando la modalità a raggi X, rileva anche attraverso ostacoli (come il Radar precedente), ma sempre solo se gli oggetti sono definiti come "Actor".

3.1.3.2 Controllori

Successivamente ai sensori, la seconda colonna presente nel Logic Editor è composta dai controllori, che come spiegato si occupano di ricevere gli output di tutti i sensori ad essi connessi e decidere se attivare o meno gli attuatori seguenti. L'impostazione generale è la stessa descritta per i sensori, con l'eccezione che la colonna dei controllori possiede un selettore degli stati, valido per tutti i mattoni logici, inclusi sensori e attuatori. Quando un sensore è attivato invia un impulso positivo e quando è disattivato uno negativo. Il compito di un controllore è quello di controllare e combinare questi impulsi per triggerare la risposta opportuna agli attuatori che sono connessi in cascata. La logica di esecuzione si basa sulle regole dell'algebra booleana oppure, se non è sufficiente, l'utente può specificare internamente a un controllore uno script Python.

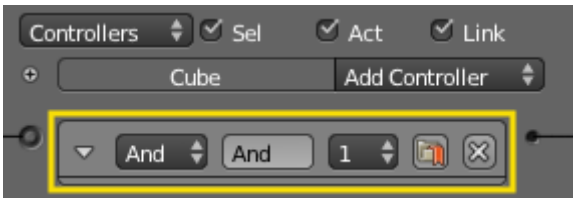


Figura 29. Opzioni comuni di un controllore.

Le opzioni comuni di un controllore (Fig.29) sono le principali viste per i sensori, ma al posto del pin c'è un selettore dello stato in cui opera questo controllore, che permette di scegliere tra 30 stati; accanto ad esso c'è un pulsante di priorità che, se selezionato, indica la volontà che questo controllore operi prima di tutti gli altri controllori non prioritari, un'opzione utile soprattutto per quegli script che si desidera siano eseguiti all'avvio.

I controllori a disposizione sono riportati di seguito.

- Operatori booleani: i primi sei tipi di controllori rappresentano gli operatori booleani AND, NAND, OR, NOR, XOR, XNOR. Il loro comportamento segue le regole della classica algebra booleana.

Sensori positivi	**Controllori**					
	AND	OR	XOR	NAND	NOR	XNOR
Nessuno	False	False	False	True	True	True
Uno, non tutti	False	True	True	True	False	False
Più di uno, non tutti	False	True	False	True	False	True
Tutti	True	True	False	False	False	True

  Se si necessita di un solo sensore, si ricorre a un controllore di tipo AND che si limita a propagare l'impulso sugli attuatori. Se invece l'AND riceve l'output di due sensori, un attuatore collegato verrà attivato solo se entrambi i sensori sono attivi. Al contrario, un controllore OR produrrà un impulso positivo solo se almeno uno dei suoi sensori è attivo.

- Expression: questo controllore valuta un'espressione scritta al suo interno dall'utente e genera un impulso positivo in output solo quando il risultato dell'espressione è positivo. L'utente può realizzare semplici espressioni logiche, utilizzando nomi di sensori, property di oggetti, numeri, operatori aritmetici (*, /, +, -, <, >, >=, <=, ==, !=), operatori booleani (AND, OR, NOT) e il costrutto sintattico "if". Per esempio si può testare il valore di una property facendo "coins" > 20 che ritorna TRUE se la property "coins" di quell'oggetto ha un valore maggiore di quello a destra dell'operatore ">".
- Python: quando nemmeno un'espressione logica è sufficiente a generare il comportamento del controllore voluto, l'utente può ricorrere al controllore Python che esegue uno script Python esistente. Si deve aggiungere un file nel Text Editor, in modo da poterlo richiamare da qui con il suo nome e se non si desidera eseguire l'intero file è possibile anche solo richiamarne un suo modulo interno. Lo script può interagire con le scene o i mattoni logici attraverso le API di Blender, rendendo quindi questo controllore capace di eseguire azioni complesse, il cui limite dipende solo dalle capacità del programmatore.

3.1.3.3 Attuatori

L'ultima colonna a destra è composta dagli attuatori, che vengono attivati quando il loro controllore a monte genera un impulso positivo. Gli attuatori si differenziano molto tra di loro nei campi e pulsanti che presentano al loro interno, dato che gestiscono azioni di vario tipo, condividendo solo le opzioni essenziali in comune (Fig.30), che corrispondono a quelle già illustrate della prima riga dei sensori.

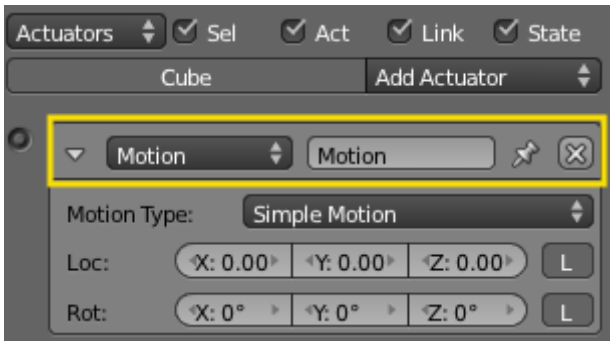


Figura 30. Opzioni comuni di un attuatore.

Attraverso gli attuatori si possono eseguire azioni quali muovere un oggetto, cambiarne la visibilità, eseguirne un'animazione, imparentare l'oggetto, creare un altro oggetto o avviare un file audio. Attualmente sono disponibili in totale ben 17 tipologie di attuatori tra cui poter scegliere, un numero elevato e in continua evoluzione, ma è utile averli ben presenti ai fini del progetto di tesi, per sapere cosa è permesso realizzare, perciò si procede qui di seguito a esaminarli brevemente:

- Action: gestisce le animazioni dell'oggetto. Permette di avviare un'animazione, scegliendo tra le varie opzioni se per esempio eseguirla una sola volta, in loop oppure una volta in avanti e poi al contrario. Si deve indicare i frame di inizio e di fine dell'animazione, in questo modo è possibile eseguirne anche solo una porzione. Presenta molte altre opzioni facoltative, come la possibilità di ricorrere a una property in cui conservare il numero del frame corrente.
- Camera: possiede opzioni per seguire in modo regolare gli oggetti. Utilizzato principalmente per permettere alle camere di seguire o tracciare un oggetto, ma può essere applicato a qualsiasi oggetto. Richiede di specificare l'oggetto che dev'essere seguito dalla camera e si può indicare un'altezza alla quale tenersi sopra di esso durante il tracciamento, una distanza minima e massima da mantenere e un valore di damping, che rappresenta la forza con cui la camera si attiene a questi vincoli.
- Constraint: vincoli che sono utilizzati per limitare le posizioni, distanze, rotazioni e campi di forza dell'oggetto, utili per controllare la fisica dell'oggetto nel gioco. In base a quale di queste

quattro modalità si seleziona, cambiano le opzioni e i campi da definire. Vincoli sulla posizione permettono di limitare la posizione dell'oggetto lungo un singolo asse, vincoli sulla distanza mantengono la distanza che l'oggetto deve avere da una superficie, vincoli sull'orientamento limitano l'asse specificato nel gioco a una direzione specifica nell'asse del mondo, mentre i vincoli del campo di forza creano una zona di campo di forza lungo un asse dell'oggetto.

- Edit Object: permette all'utente di modificare le impostazioni degli oggetti nel gioco. Le opzioni incluse nel menu consentono di: aggiungere un oggetto scegliendolo tra quelli della scena corrente e posizionandolo al centro di questo, eliminare questo oggetto, sostituire la mesh dell'oggetto con un'altra e volendo anche la sua fisica, far seguire un altro oggetto, impostare opzioni di dinamica all'oggetto.
- Filter 2D: numerosi filtri per effetti speciali come colori seppia, scala di grigi, erosione, sfocatura, motion blur, filtro laplaciano, filtro di Sobel e altri. Per tutti questi filtri è richiesto di specificare un solo parametro indicante il numero di passaggi con cui applicarlo. Inoltre è possibile realizzare filtri personalizzati utilizzando il Text Editor per scrivere il proprio filtro e richiamando in questo attuatore il nome del proprio script.
- Game: gestisce l'intero gioco, permettendo di riavviare il gioco, chiuderlo, salvarlo, caricarlo o caricarne un altro specificando un file.
- Message: questo attuatore è la controparte del sensore Message visto in precedenza. L'attuatore Message si occupa di inviare dati a qualsiasi oggetto desiderato e il sensore omonimo serve per stare in ascolto di questi messaggi e attivarsi quando li riceve. L'attuatore presenta un campo "To" in cui specificare l'oggetto a cui destinare il messaggio, inviabile in broadcast a tutti gli oggetti se non si indica un destinatario. Invece il campo "Subject" consente di filtrare tra i messaggi inviati, in questo modo un oggetto che riceve più messaggi può possedere un sensore che si attiva solo con messaggi aventi un particolare soggetto, riferito a uno scopo di gioco specifico. Infine il campo "Body" permette di includere nel messaggio un testo o una property a piacere, che può essere leggibile solo da Python.
- Mouse: gestisce la visibilità e la sensibilità dei movimenti del mouse.
- Motion: realizza la dinamica degli oggetti. In modalità Simple Motion consente di spostare l'oggetto di posizione o ruotarlo istantaneamente, ma così non passa attraverso i punti intermedi e può interferire con le simulazioni fisiche di altri oggetti, portando un oggetto a passare attraverso un altro. Invece in modalità Servo Control si può impostare una velocità obiettivo e specificare quanto in fretta debba raggiungere tale velocità, in questo modo produce velocità fisicamente corrette e lascia l'aggiornamento della posizione alla simulazione fisica.
- Parent: imposta un padre all'oggetto o gli rimuove l'imparentamento.

- Property: manipola le property dell'oggetto. Permette di impostare un valore desiderato a una property, sommarci un valore a quello già presente o copiare in una property il valore presente in quella di un altro oggetto.
- Random: crea un valore random memorizzabile in una property, immettendo un seme.
- Scene: gestisce le scene nel file Blender. Queste possono essere utilizzate come livelli o per interfaccia utente e sfondo. L'attuatore permette di riavviare la scena corrente resettandone il contenuto, cambiare la scena passando a quella desiderata, cambiare la camera utilizzata, aggiungere una scena sovrapposta alla corrente fungendo da overlay e di solito utile come interfaccia, aggiungere una scena di sfondo, rimuovere una scena, sospendere la scena attuale o riprenderla.
- Sound: utilizzato per avviare suoni nel gioco. Si sceglie un file audio da disco e si gestisce il volume e altri aspetti della riproduzione.
- State: Gestisce gli stati degli oggetti. Permette di cambiare lo stato attuale con un altro, rimuovere stati da quelli attivi, aggiungere stati a quelli attivi o settare uno stato attivo.
- Steering: permette di far seguire un percorso. Muove un oggetto verso un oggetto obiettivo, con varie opzioni settabili per gestire lo spostamento, quali accelerazione massima, velocità, distanza massima e altri.
- Visibility: cambia la visibilità dell'oggetto. Oltre a modificare la visibilità dell'oggetto corrente, permette di attivare o meno l'occlusione e di applicare in cascata su tutti i figli dell'oggetto questi settaggi.

3.2 La teca olografica

Le scene interattive che si intende permettere di realizzare sono pensate per essere visualizzate, ad esempio, all'interno di una teca olografica che era stata progettata e prodotta nell'ambito di un precedente lavoro di tesi per il corso di laurea in Design e Comunicazione Visiva (Fig.31) [54]. Tale struttura espositiva consiste in una teca piramidale posta su una base di circa 83x67cm e sormontata da un televisore 32" rivolto verso il basso, che si occupa di proiettare l'immagine da visualizzare al suo interno. Si tratta di una soluzione abbastanza economica, modulare e trasportabile per la realizzazione di ologrammi, che permette un buon risultato visivo. Frontalmente ad essa è presente un alloggiamento integrato per ospitare il Leap Motion, che permette di rilevare i gesti manuali realizzati dalla persona che si trova davanti. Il fatto che la teca sia visibile anche lateralmente permette a più persone di visualizzarne il contenuto in contemporanea, quando dinanzi vi è un utente che ci sta interagendo.

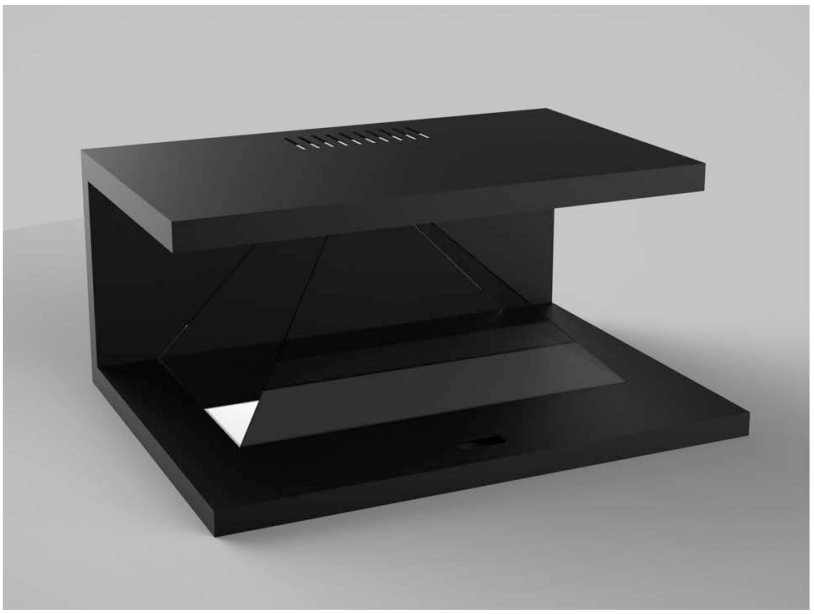


Figura 31. Un render del progetto della teca. (fonte: tesi di Laurea in Design e Comunicazione Visiva di Laura Venturi)

La proiezione delle immagini avviene sfruttando il principio del Pepper's Ghost, presentato per la prima volta nel 1863 da John Herny Pepper durante l'esposizione teatrale del "The Haunted Man" di Charles Dickens. Infatti questa illusione venne utilizzata agli albori soprattutto in ambito teatrale, dove permetteva per esempio di simulare sul palco la presenza di personaggi "fantasma", trovando impiego anche in campo cinematografico e di intrattenimento.

Come schematizzato in Figura 32, l'effetto consiste della proiezione di un'immagine, da una fonte non visibile all'osservatore, che viene riflessa attraverso un vetro inclinato a 45 gradi rispetto allo spettatore, creando l'illusione di osservare frontalmente un oggetto virtuale. Nel caso della teca piramidale impiegata, si ottiene una proiezione dell'oggetto visualizzato dal monitor sovrastante, dando la percezione che l'oggetto sia localizzato su un piano al suo interno. Affinché l'ologramma generato sia coerente anche osservandolo lateralmente, l'immagine generata dallo schermo deve essere scomposta in tre inquadrature per le rispettive angolazioni di vista. La scena sarà pertanto inquadrata da tre camere, una frontale e due laterali, calibrate opportunamente in modo da generare una resa finale ottimale anche se l'utente si sposta attorno alla teca. Naturalmente l'area interessata dalla proiezione non corrisponderà a tutto il volume della piramide, ma solo a quella porzione centrale della piramide in comune tra le tre viste.

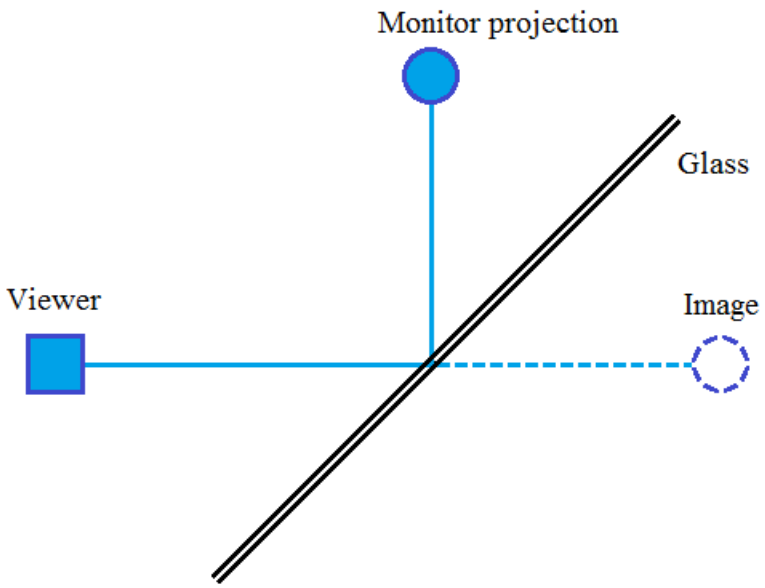


Figura 32. Schema dell'illusione Pepper's Ghost.

3.3 Leap Motion

Come si ha anticipato, per interagire con la teca olografica illustrata si ha scelto di ricorrere al riconoscimento di gesti manuali per mezzo del Leap Motion. Il Leap Motion è una piccola periferica USB per il motion control, progettata per essere posizionata su una scrivania e rivolta verso l'alto. Alternativamente può anche essere montata su un visore per la realtà virtuale. Attraverso l'uso di telecamere e LED infrarossi, il Leap è in grado di riconoscere i gesti delle mani realizzati nel suo campo visivo, tracciando i movimenti delle dita con una elevata precisione. Il dispositivo supporta movimenti di mani e dita come input, ma senza richiedere un contatto con le mani o tocchi e si prefigge lo scopo di sostituire il mouse del computer per eseguire le proprie azioni [55].

La tecnologia del Leap Motion è stata sviluppata inizialmente nel 2008 da David Holz, che il 1 novembre 2010 fonda la Leap Motion, Inc. con sede in California e inizialmente sotto il nome di OcuSpec, assieme all'amico Michael Buckwald. Raccolti i fondi necessari grazie a vari investimenti di capitali, il 21 maggio 2012 la società annuncia il suo primo prodotto, chiamato inizialmente The Leap, il cui lancio commerciale avviene nel 2013 a un prezzo di 79,99 dollari. All'inizio le aspettative della società erano elevate, prevedendo di stravolgere il modo in cui gli utenti interagiscono con il proprio sistema operativo o navigano in internet. Nel 2014 viene rilasciata una seconda versione del software e anche una modalità di tracking per la realtà virtuale quando il dispositivo è montato su un visore come ad esempio l'Oculus Rift. Nel febbraio 2016 è stato distribuito un nuovo software chiamato Orion, progettato specificatamente per la VR.

Nel campo del motion tracking, la console Wii diffuse l'interazione gestuale, mentre il Kinect di Microsoft ha liberato l'utente dalla necessità di impugnare un controller, ampliando il tracciamento dei movimenti a tutto il corpo [56]. Il Leap, rispetto al Kinect, si differenzia nel focalizzarsi solamente sulle mani piuttosto che l'intero corpo e osservando un'area più ristretta, ottenendo così una maggiore precisione dei gesti producibili da mani e dita. Per questi motivi lo si è preferito agli altri dispositivi di gesture recognition, unito alle sue ridotte dimensioni.

Figura 33. Il Leap.

Il Leap Motion (Fig.33) è un dispositivo rettangolare di piccole dimensioni, comodamente trasportabile e sufficientemente leggero per essere eventualmente montato su un visore [57]. Misura circa 76x30mm per appena 13mm di spessore e pesa sui 45g, con una presa USB per collegarlo a computer. Al suo interno incorpora due telecamere a infrarossi monocromatiche e tre LED a infrarossi con lunghezza d'onda di 850nm, con cui osservare una zona sopra di esso che assume una forma più o meno emisferica. L'area di interazione coperta (Fig.34) arrivava inizialmente a circa 60cm sopra il controller, per 60cm di larghezza e profondità su ogni lato e un ampio angolo di vista di 150° in larghezza e 120° in profondità. Con il software Orion si è successivamente aumentato questo range visivo a 80cm nelle varie direzioni. Questa distanza è limitata dalla propagazione della luce LED nello spazio, siccome diventa molto più difficile dedurre la posizione delle mani nell'ambiente 3D oltre a una certa distanza. Invece l'intensità della luce LED è attualmente limitata dalla corrente massima che può essere inviata sulla connessione USB.

Figura 34. Area di interazione del Leap Motion collegato a un computer. (fonte: www.leapmotion.com)

I LED generano un modello 3D di punti di luce IR e le telecamere generano quasi 300 fotogrammi al secondo di dati riflessi, che vengono poi inviati attraverso il cavo USB al computer, dove il software del controller analizza tali dati confrontando i fotogrammi 2D generati dalle due telecamere. I dati prendono la forma di un'immagine stereo in scala di grigi dello spettro di luce vicino all'infrarosso, separati tra camera sinistra e destra. Solitamente saranno visibili i soli oggetti direttamente illuminati dai LED del controller, tuttavia lampadine e luce solare illumineranno anche la scena in infrarosso. Una volta che i dati delle immagini sono trasmessi via cavo USB al computer, si procede a elaborarle: il controller non genera una mappa di profondità, ma applica algoritmi avanzati ai dati grezzi del sensore. Il software Leap Motion Service processa le immagini, compensando gli oggetti sullo sfondo e le luci ambientali, e ricostruisce una rappresentazione 3D di cosa osserva il dispositivo. Affinché il riconoscimento e il tracciamento funzionino al meglio, serve che il controller possieda un'immagine chiara delle silhouette degli oggetti, ad alto contrasto. Successivamente si combinano i dati con un modello interno della mano umana e si estraggono le informazioni di tracciamento, distinguendo le dita e gli strumenti. Gli algoritmi di tracciamento interpretano i dati 3D e deducono le posizioni degli oggetti coperti alla vista, mentre vengono applicate delle tecniche di filtraggio per garantire una coerenza temporale regolare dei dati. Infine il Leap Motion Service genera i risultati, espressi in forma di serie di frame e contenenti tutti i dati di tracciamento, in un protocollo di trasporto. Attraverso questo protocollo, il servizio comunica con il Leap Motion Control Panel attraverso una connessione socket locale. L'SDK fornisce due tipi di API per accedere ai dati di tracciamento: un'interfaccia nativa e una WebSocket. La libreria client organizza i dati in una struttura API orientata a oggetti, gestisce la cronologia dei frame e fornisce

funzioni di aiuto e classi. Da qui, la logica dell'applicazione si lega all'input del Leap Motion permettendo un'esperienza interattiva con il motion control.
Le informazioni di posizione sono misurate con un sistema di coordinate sinistrorso centrato sul Leap e con una precisione di 0,01mm. Si hanno a disposizione anche altri tre sistemi di coordinate: uno riferito allo schermo e con l'origine nell'angolo in basso a sinistra dello schermo, uno relativo alla "scatola di interazione" che corrisponde a uno spazio sopra al Leap interno al volume di vista e uno basato sulla zona di contatto utilizzabile per emulare un'interazione su uno schermo immaginario. Invece le informazioni ricavabili da una mano riguardano la velocità del palmo e la sua velocità vettoriale rispetto alle dita, mentre la rotazione si può esprimere in forma matriciale oppure in termini di inclinazione, rollio e imbardata. Infine si può avere una valutazione della curvatura della mano, che si ricava simulando una sfera appoggiata sul palmo e calcolando il raggio di tale sfera, che diminuisce proporzionalmente alla chiusura delle dita della mano.
Il Leap è in grado di riconoscere le varie dita, distinguendo le due mani, gli oggetti impugnati, come una penna e eventuali gesti effettuati. Grazie a un tale controllo, si possono ottenere varie tipologie di applicazioni, da giochi basati su una precisa interazione delle mani a strumenti musicali. Per un dito si può determinare la sua larghezza e lunghezza, la posizione della punta, la direzione e la velocità di movimento. Solitamente una penna è riconoscibile in quanto più sottile, lunga e dritta rispetto a un dito.
Ottenendo i vari frame sequenziali, si può analizzare l'evoluzione temporale del tracciamento di mani e dita per riconoscere i gesti effettuati dall'utente. Certi pattern di movimento possono essere considerati come dei gesti in quanto potrebbero indicare un intento o un comando da parte dell'utente, perciò il software del Leap segnala tali gesti e per ogni gesto osservato, aggiunge un oggetto Gesture al frame. Se il gesto continua nel tempo, il software aggiunge oggetti Gesture aggiornati sui frame successivi, che fanno riferimento allo stesso gesto in corso in quanto condividono lo stesso valore ID.
L'API fornita permette di riconoscere le seguenti quattro tipologie base di gesture: Circle, Swipe, KeyTap e ScreenTap. Eventualmente si potrebbe estendere questo insieme di gesture nativamente riconoscibili dal dispositivo per mezzo di personali applicazioni basate sull'utilizzo di algoritmi di tipo DTW (Dynamic Time Warping). Qui di seguito si illustra un'analisi dei quattro gesti disponibili.

- Circle: movimento di un singolo dito che traccia un cerchio nello spazio. Si tratta di un gesto che si protrae nel tempo, perciò il suo progresso viene aggiornato lungo i frame, finché il gesto non termina. Un Circle è considerato terminato quando il dito o strumento che lo eseguiva termina il movimento circolare o va troppo piano. Un Circle può essere realizzato con qualsiasi dito e anche con uno strumento, ma solitamente è realizzato con l'indice e le altre dita della mano chiuse, come in Figura 35. Attraverso i metodi messi a disposizione, si possono distinguere due tipi di Circle, orario e antiorario, in base al verso in cui si realizza il cerchio e inoltre si può conoscere il raggio del cerchio tracciato.

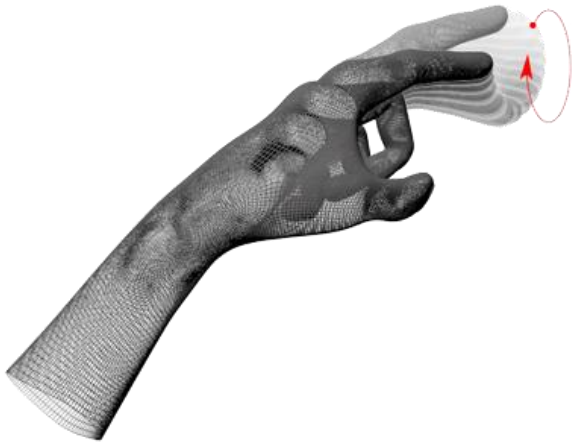


Figura 35. Gesto Circle effettuato con l'indice. (fonte: www.leapmotion.com)

- Swipe: un movimento lineare e ampio di un dito (Fig.36). Anche questo gesto continua nel tempo e dunque avrà un oggetto Gesture che si aggiorna nei vari frame. In questo caso, il dito compie un movimento lineare che prosegue fino a quando il dito cambia direzione o si sposta troppo lentamente. Una Swipe può essere distinguibile in orizzontale, rivolta da sinistra verso destra o viceversa, e verticale, dall'alto verso il basso o viceversa, quindi in totale si possono definire quattro tipi essenziali di Swipe. Anche la lunghezza della Swipe è uno dei vari parametri che vengono memorizzati.

Figura 36. Gesto Swipe orizzontale. (fonte: www.leapmotion.com)

- Key Tap: un movimento di tapping di un dito verso il basso, come se si stesse premendo un tasto della tastiera (Fig.37). Un Key Tap viene riconosciuto quando si rileva che un dito si abbassa, per poi risalire, mentre il resto della mano resta fermo. I gesti di tipo Tap, a differenza di Circle e Swipe, sono discreti nel tempo e quindi il Leap Motion software ne genera un solo oggetto Gesture.

Figura 37. Un Key Tap realizzato con l'indice. (fonte: www.leapmotion.com)

- Screen Tap: un movimento di tapping di un dito simile al precedente, ma in questo caso è fatto in avanti, come se si stesse toccando lo schermo verticale di un computer (Fig.38). Si effettua con un dito teso in avanti e l'intera mano avanza in modo relativamente veloce. Per i gesti Tap

una delle informazioni recuperabili è la velocità applicata, ma nel caso dello Screen Tap è anche possibile conoscere le coordinate del punto in cui è stato rilevato.

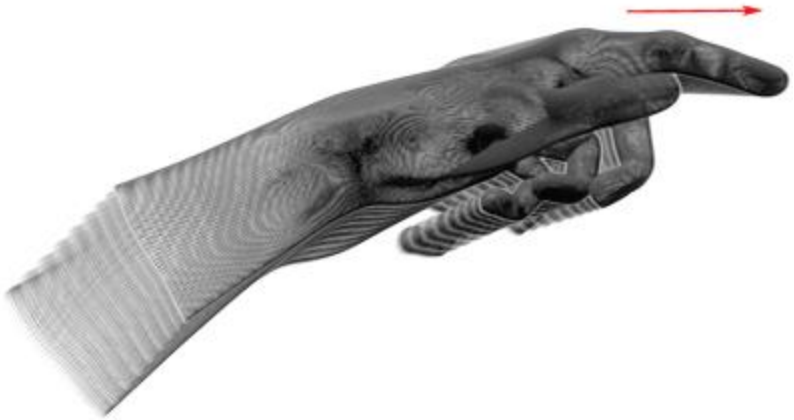


Figura 38. Uno Screen Tap realizzato con l'indice. (fonte: www.leapmotion.com)

3.4 JavaFX

Per realizzare l'applicazione Visual Scene Editor si ha scelto di utilizzare il linguaggio di programmazione JavaFX [58]. JavaFX è una libreria grafica cross-platform basata sulla piattaforma Java, per la creazione sia di applicazioni desktop stand-alone, sia di Rich Internet Application (RIA), su Microsoft Windows, Linux e macOS. Per programmare, si può ricorrere all'IDE NetBeans, un ambiente di sviluppo integrato con la libreria JavaFX, oppure con Eclipse, attraverso il plugin "e(fx)clipse".
JavaFX è il successore di Swing, che lo sostituisce come libreria GUI standard per il Java SE [59]. La prima versione risale al 2008, distribuita dalla Sun Microsystems e prima della versione 2.0 del 2011 utilizzava un linguaggio JavaFX Script per creare le applicazioni JavaFX, invece le versioni successive sono implementate come libreria Java nativa e le applicazioni che utilizzano JavaFX sono scritte in codice Java "nativo". Nel 2014 JavaFX è giunto alla versione 8, in quanto segue la numerazione di Java 8. Infatti ora la libreria grafica JavaFX è inclusa in Java SE, con le API JavaFX [60] integrate nel JDK Java [61], e a partire da Java 8 è stato introdotto il supporto alle architetture ARM, molto diffuse in ambito mobile. La versione JavaFX 9 è in procinto di essere rilasciata.
Nel realizzare un'applicazione con JavaFX ci si basa essenzialmente sui componenti chiave Stage, Scene e Node (Fig.39). Lo Stage rappresenta l'intera finestra applicativa, che fa da contenitore principale, mentre la Scene è l'effettiva interfaccia con cui l'utente interagisce, perciò se ne può rappresentare solo una per volta sullo Stage in un determinato istante, ma può avvicendarsi a un'altra e

funge da contenitore dei nodi, infine un Node è un singolo elemento della Scene, dotato di un suo aspetto visivo e di un comportamento. Un nodo può includere oggetti 2D o primitive 3D come box, cilindri o sfere, controlli UI come pulsanti, label o aree di testo, pannelli di layout, grafici ed elementi media quali audio, video o immagini (Fig.40). Una Scene contiene uno "scene graph", che è una collezione di tutti gli elementi che compongono l'interfaccia utente e che derivano tutti dalla classe Node. I nodi sono disposti in una gerarchia, dove ciascun nodo della Scene ha un solo genitore e può avere uno o più figli. Inoltre il componente WebView di JavaFX permette di incorporare HTML in una Scene, utilizzando il motore di rendering Web Kit, che offre il supporto a HTML5, CSS, JavaScript, DOM e SVG.

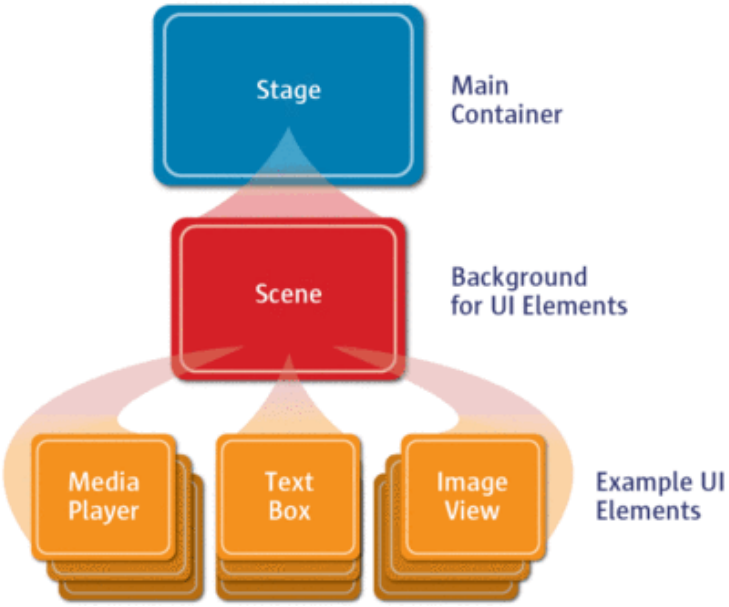


Figura 39. Struttura di un'applicazione JavaFX. (fonte: www.oracle.com)

JavaFX usa la gestione degli eventi per rispondere a eventi di input dell'utente, associandoci la dipendenza dalle proprietà. JavaFX supporta i concetti di proprietà e usa le proprietà in modo estensivo nelle sue classi. In breve, una proprietà è una variabile il cui valore può essere osservato. Si può registrare un listener con qualsiasi proprietà, con la possibilità di scrivere codice che viene attivato automaticamente ogni volta che la proprietà cambia. Per esempio, si può agganciare un event listener al colore di una shape: se la shape cambia colore, il codice del relativo event listener viene eseguito. In aggiunta, si possono collegare ("bind") proprietà tra di loro, in modo che se una proprietà cambia di valore, il valore dell'altra proprietà cambia con esso.

Una delle migliori caratteristiche di JavaFX è che si può controllare la formattazione con CSS (Cascade Style Sheets). L'uso di file CSS consente di personalizzare lo stile degli elementi che compongono il layout dell'applicazione, per applicare effetti grafici a pannelli, aree di testo o pulsanti. Pressoché ogni

aspetto dell'interfaccia utente può essere settato da una regola di stile e si può facilmente permettere all'utente di selezionare quale dei vari style sheet disponibili applicare alla Scene. Così si può cambiare l'intera visualizzazione dell'applicazione con la chiamata di un singolo metodo. JavaFX permette di applicare animazioni ed effetti speciali a ogni Node nello scene graph, come shadow, reflection, blur e altri interessanti effetti visivi che possono trasformare l'aspetto dell'interfaccia utente. Infine JavaFX presenta il supporto per i comuni gesti touch come lo scrolling, swiping, rotating e zooming. Gestire questi eventi in JavaFX è facile tanto quanto gestire ogni altro tipo di eventi: si deve semplicemente installare un event listener sull'evento touch e poi scrivere il codice che risponde in modo appropriato.

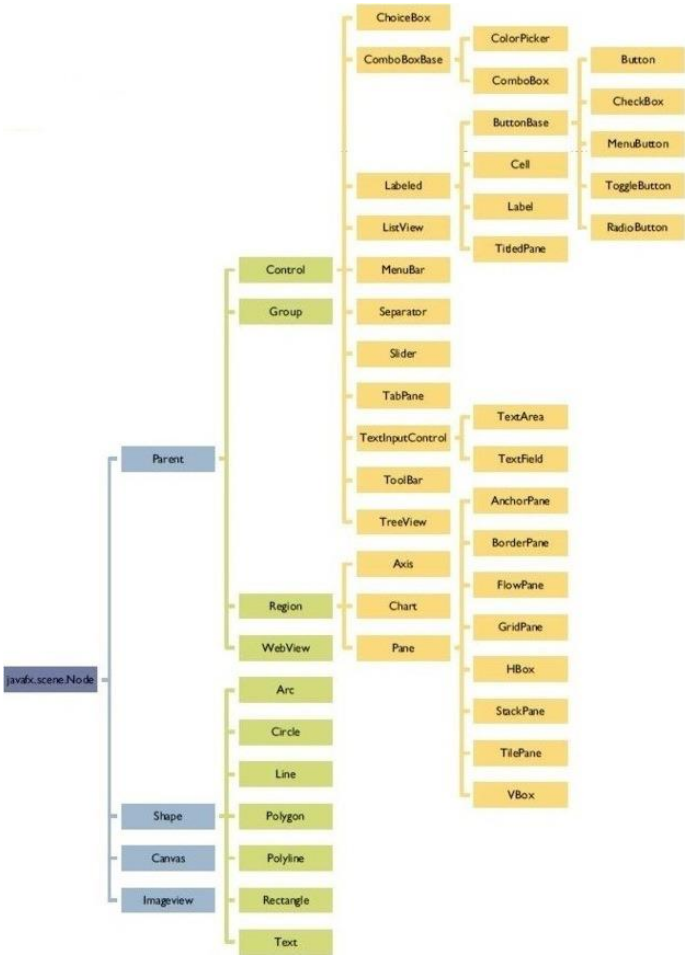


Figura 40. Albero delle principali componenti di JavaFX. (fonte: www.gianlucadivincenzo.it)

3.4.1 Scene Builder

Per aggiungere elementi visivi in JavaFX si può operare da codice, ma risulta poco immediato. Per questo motivo, è preferibile definire la disposizione dei componenti grafici attraverso il tool JavaFX

Scene Builder fornito da Gluon [62] (Fig.41), perché Oracle lo rende disponibile solo più sotto forma di codice sorgente.

Un aspetto interessante di JavaFX è la facilità con cui viene implementato il pattern MVC (Model-View-Controller). Con lo Scene Builder si può definire la GUI all'interno di un particolare file XML, scritto in linguaggio FXML. L'uso dell'FXML permette di separare la progettazione grafica dalla logica funzionale che è definita nel codice di programmazione e lo Scene Builder si occupa di generare in automatico un file FXML costruendo visualmente l'interfaccia grafica, in modo molto semplice. Il FXMLoader è un apposito oggetto di JavaFX che connette un'interfaccia FXML a un Controller che servirà da ricettore di tutti gli eventi innescati dall'utente.

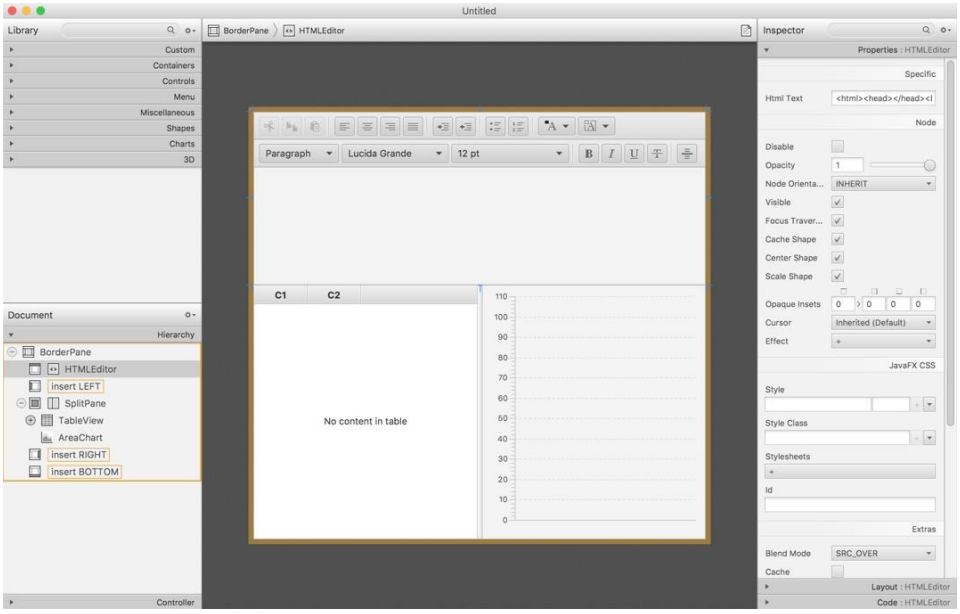


Figura 41. Gluon Scene Builder.

Lo Scene Builder presenta un'interfaccia utente Drag & Drop intuitiva, che permette un'interazione rapida. L'utente ha accesso a un'ampia libreria di elementi grafici disponibili per realizzare l'interfaccia, suddivisi in categorie. Selezionato l'elemento desiderato, lo si trascina nella finestra al centro posizionandolo dove si desidera inserirlo oppure si trascina nella colonna della gerarchia, per indicare a quale padre agganciarlo. La colonna di destra permette di intervenire sulle proprietà dell'elemento attualmente attivo, modificarne il layout e dichiarare i metodi associati a esso che saranno definiti nel proprio codice, tutte azioni eventualmente realizzabili anche direttamente sul file FXML, ma rese rapide e intuitive dallo Scene Builder.

Capitolo 4

Progetto

Illustrato lo stato dell'arte esistente e le tecnologie impiegate, si può passare a trattare il progetto effettivo del lavoro di tesi. Si descriverà pertanto come è stata realizzata l'applicazione Visual Scene Editor e come essa permette all'utente di definire la logica di interazione ad alto livello in modo intuitivo, per poi analizzare come il proprio risultato prodotto viene importato in Blender, attraverso un add-on che legge da file la simulazione desiderata e la replica nel Logic Editor di Blender.

4.1 L'applicazione Visual Scene Editor

Nel realizzare l'applicazione, il precedente programma Leap Embedder esistente è stato preso in considerazione come modello di riferimento a proposito di quali fossero i concetti essenziali da re-implementare e quali fossero invece gli aspetti che non convincevano di quella implementazione. Tenendo in considerazione lo stato dell'arte esistente riguardo ai linguaggi di programmazione visuali, si sono abbozzate varie anteprime del programma "ideale", fino a che non si ha scelto come procedere a realizzarlo. Non sono mancati aggiustamenti secondari in corso d'opera fino alla versione finale che si descriverà nei vari componenti.

4.1.1 Interfaccia dell'applicazione preesistente

Prima di procedere a illustrare l'interfaccia realizzata per il Visual Scene Editor che si rivela essere fortemente scene-centric, ovvero pone in risalto le scene e le loro interazioni, si mostra una veloce panoramica del precedente programma, maggiormente object-centric. Come mostrato in Figura 42, l'interfaccia dell'applicazione Leap Embedder si compone di quattro aree distinte. La prima è la toolbar in alto, dove si hanno i vari pulsanti necessari per la gestione del progetto. Sulla destra possiede una lista dei nomi degli oggetti inclusi nel file Blender e ciascuno di essi presenta un tab nel pannello centrale, dove se ne definisce la logica. Sulla sinistra si hanno gli strumenti con cui definire tale logica di interazione, come creare o rimuovere Gesture e Action, definire un oggetto quale puntatore nella scena o impostare le scene in cui è presente. Infine, al centro c'è l'area principale di gestione della logica di

interazione, dove si interagisce con un solo oggetto alla volta. L'impostazione non si discosta molto da quella del Blender Game Engine, in quanto i sensori vengono sostituiti dalle Gesture e gli attuatori dalle Action, ma per il resto il funzionamento è piuttosto simile.

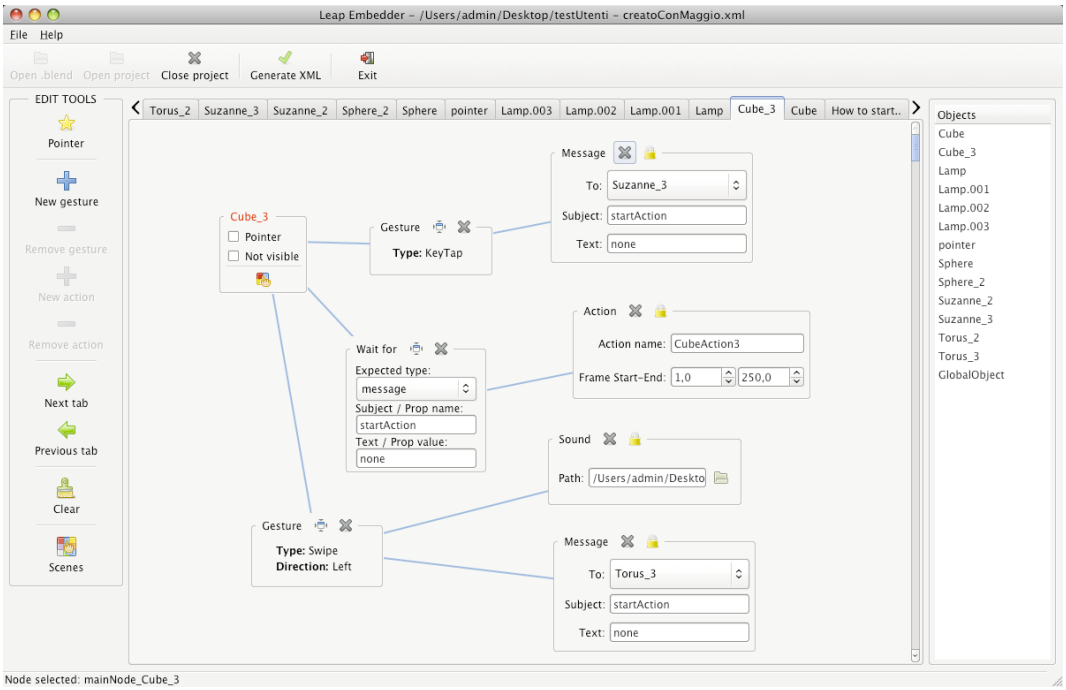


Figura 42. Interfaccia del precedente Leap Embedder.

Ogni oggetto ha una sua scheda in cui c'è un nodo che lo rappresenta, con indicato le sue proprietà di puntatore e visibilità. A questo nodo oggetto si possono collegare più nodi di tipo Gesture, ai quali possono a loro volta essere agganciati più nodi di tipo Action. I nodi Gesture rappresentano i gesti rilevabili con il Leap Motion, più un nodo WaitFor che equivale al sensore di tipo Message del BGE, ovvero resta in attesa di ricevere un messaggio. Selezionato un nodo Gesture, si può aggiungere in cascata uno o più nodi Action che corrispondono agli attuatori del BGE: si può dichiarare per esempio un nodo Action che scateni un'animazione, un cambio di scena o appunto un messaggio che sarà poi rilevato da un opportuno WaitFor. Le Action definibili non coprono ovviamente tutti i possibili attuatori del BGE esistenti, ma solo quelli ritenuti più utili per lo scopo che si prefigge l'applicazione. Senza addentrarsi troppo nei dettagli, basta segnalare che i nodi Action replicano fedelmente i rispettivi attuatori che rappresentano. Per esempio, si ha un Sound per un effetto sonoro, una Action dove indicare il nome dell'animazione da scatenare e il relativo intervallo di frame oppure vi è il nodo Message che richiede di indicare in campi testuali al suo interno il destinatario del messaggio (o più di uno se si ricorre a una Action di tipo MultiMessage), un soggetto del messaggio con cui poter effettuare una distinzione dei messaggi in ricezione e un campo di testo facoltativo. Dato che il programma permette di visualizzare la scheda di un solo oggetto per volta, se si desidera che un gesto su un primo scateni

un’azione su un secondo, si deve appunto ricorrere a una coppia di nodi Message e WaitFor (per l’invio e la ricezione), esattamente come avverrebbe nel BGE. Questa implementazione rigorosa della struttura del BGE si è rivelata essere la causa principale per cui il Leap Embedder precedente non avesse le basi per raggiungere l’obiettivo di rivelarsi semplice e intuitivo per un utente comune.

Oltre a possedere una scheda per ogni oggetto, si ha una scheda riservata al cosiddetto “oggetto globale”. Questo “GlobalObject” è stato aggiunto per permettere di modellare delle azioni che si vuole che si verifichino indipendentemente da quale sia la scena attualmente impostata. Se ad esempio si desiderasse che quando viene rilevata una Gesture di tipo Circle la simulazione ricarichi la scena iniziale, indipendentemente dalla scena in cui ci si trova, si definisce tale logica in questo oggetto globale, che provvederà a replicarla in tutte le scene una volta che si reimporterà il risultato finale in Blender. Infine esiste un’altra scheda rivolta alla definizione delle scene (Fig.43) ed è quella che viene visualizzata per prima all’apertura di un file Blender. Da questa scheda apposita si creano le scene desiderate e per ognuna si ha una lista di checkbox per tutti gli oggetti esistenti, con cui settare quali oggetti includere in essa, mentre la checkbox “Start scene” a inizio lista indica se quella debba essere la scena di partenza della simulazione, siccome è richiesto che esista una scena settata come tale, che verrà avviata per prima.

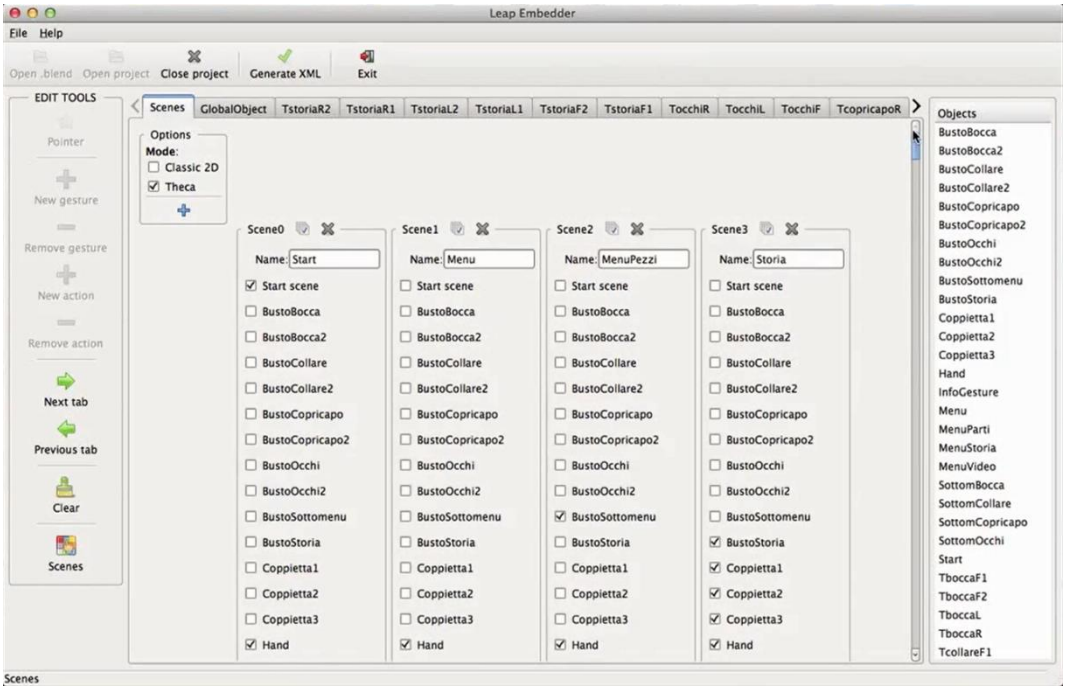


Figura 43. Scheda di gestione delle scene.

4.1.2 Progettazione interfaccia

Secondo la norma ISO 9241-11, la definizione di usabilità di un prodotto è "l'efficacia, l'efficienza e la soddisfazione con cui utenti specifici possono raggiungere determinati obiettivi in particolari contesti" [63]. Pertanto, per l'International Standard Organization (ISO), l'usabilità non è una semplice caratteristica di un prodotto, ma dipende anche da chi deve utilizzare tale prodotto, dai risultati che si vogliono conseguire e dall'ambiente nel quale il prodotto viene utilizzato [64].
Nel realizzare la nuova applicazione, per favorire la migliore esperienza utente possibile, si è cercato di seguire una nota valutazione euristica teorizzata da Jakob Nielsen [65], che si basa su un metodo di valutazione dell'usabilità di un applicativo analizzandone l'interfaccia secondo 10 principi universali o linee guida [66] derivanti dall'applicazione di tecniche di analisi fattoriale su un ampio spettro di problemi di usabilità [67]. Le cosiddette 10 euristiche di Nielsen che è raccomandabile seguire sono le seguenti.

1. Visibilità dello stato del sistema: l'applicazione dovrebbe sempre mantenere gli utenti aggiornati sullo stato delle loro azioni, fornendo un adeguato feedback in un tempo ragionevole.
2. Corrispondenza tra sistema e mondo reale: l'applicazione dovrebbe parlare il "linguaggio" degli utilizzatori finali, ricorrendo a parole, frasi e concetti a loro familiari (per esempio, azioni comuni come "salva con nome" o "copia e incolla", un'icona "cestino", ecc.).
3. Controllo e libertà: l'utente dovrebbe avere il controllo sulle informazioni visualizzate (evitando azioni non volute) e muoversi liberamente tra i vari argomenti presenti, senza percorsi predefiniti, ma secondo le varie esigenze, anche fornendo delle scorciatoie.
4. Consistenza: l'utente deve aspettarsi che le convenzioni dell'applicazione siano valide per tutta l'interfaccia, mantenendo la sensazione di essere sempre nello stesso ambiente, grazie anche alla presenza di elementi grafici ricorrenti.
5. Prevenzione dell'errore: occorre cercare di prevenire che l'utente incorra in un errore, assicurando comunque la possibilità di correggersi e tornare allo stato precedente.
6. Riconoscimento piuttosto che ricordo: si consiglia di produrre layout semplici e schematici, per facilitare la consultazione delle informazioni, con gli elementi essenziali sempre chiari e visibili. Evitare di richiedere all'utente di ricordare il posizionamento e funzionamento degli oggetti.
7. Flessibilità ed efficienza d'uso: converrebbe fornire la possibilità agli utenti di un uso differenziale dell'interfaccia a seconda della sua esperienza, per esempio includendo scorciatoie (acceleratori) utili per gli utenti esperti.
8. Estetica e progettazione minimalista: visualizzare in primo piano tutte e sole le informazioni necessarie, facendo prevalere il contenuto all'estetica. Non far risaltare oggetti irrilevanti o raramente necessari e non confondere o distrarre l'utente.

9. Aiutare l'utente a riconoscere, diagnosticare e correggere gli errori: fornire messaggi di errore comprensibili e quanto più precisi possibile sul problema, per suggerire un'eventuale soluzione costruttiva. Chiedere conferma alle azioni importanti e segnalare situazioni di conflitto o di informazioni incomplete.
10. Documentazione: è preferibile che l'applicazione si possa utilizzare senza documentazione, comunque fornendo aiuto e documentazione, facilmente individuabile, evidenziando le attività eseguibili dall'utente e mantenendo dimensioni contenute.

Per massimizzare l'usabilità di un'applicazione esistono diversi modelli che fanno riferimento essenzialmente a due principali modalità di funzionamento cognitivo. La prima è l'inferenza, ovvero si valuta la complessità dell'applicazione in base al numero di azioni necessarie e del numero di concetti e procedure da ricordare per realizzare l'interazione richiesta. In questo contesto vengono in aiuto l'impiego di menu e di tasti funzione. Il secondo modello è quello analogico, dove si prende in considerazione la similitudine con altre conoscenze possedute dall'utente, riducendo la complessità dell'interazione attraverso l'uso di metafore che generano un'associazione a un contesto più familiare all'utente [68].

Prima di iniziare a creare l'interfaccia grafica si sono abbozzati degli schizzi preparatori, i cosiddetti mockup, con cui farsi un'idea della validità dell'impostazione che si desiderava adottare, per poi procedere effettivamente a realizzare quella ritenuta migliore. Si sono analizzate molteplici impostazioni dell'interfaccia, talvolta scartate velocemente, in altri casi accorgendosi della loro scomodità di utilizzo o poca intuitività solo dopo una prima realizzazione.

Nel progettare l'interfaccia del programma si è tenuto conto dei pregi e dei difetti dell'applicazione presa a riferimento, degli obiettivi prefissati e delle analisi effettuate nel capitolo 2 sui linguaggi di programmazione visuali, cercando di riprendere le idee e i concetti più utili per le specifiche necessità.

Ai fini dell'utilizzo richiesto dallo strumento la cui realizzazione è oggetto del lavoro di tesi, le considerazioni di fondo sono che si deve poter accedere a una lista di tutti gli oggetti a disposizione, in modo da poter avere una visione generale del contenuto del file Blender e selezionare quelli che si desidera inserire nelle varie scene. Di conseguenza è utile avere la possibilità di visualizzare una lista o visione d'insieme di tutte le scene generate. Un altro aspetto importante è che ogni oggetto aggiunto a una scena deve essere personalizzabile nelle interazioni possedute; si deve pertanto avere uno spazio dove poter definire i suoi comportamenti.

Il primo punto fondamentale è scegliere come rappresentare al meglio i blocchi logici. Nel Blender Game Engine si hanno sensori, controllori e attuatori collegati tra loro, mentre nel programma Leap Embedder si creano dei blocchi in cascata, anche qui da collegare, non particolarmente diversi da Blender. Se si osserva la logica di interazione alla base del BGE, si nota come il tutto possa essere riconducibile a un concetto di causa e azione, esattamente come in Kodu e Project Spark. Questa

osservazione ha portato a decidere di implementare anche nel programma desiderato l'idea del When-Do vista nei due programmi analizzati, concetto che si ripete in forme simili in molti linguaggi di programmazione visuale e che ben si presta a essere ripreso per il progetto di tesi, con gli opportuni accorgimenti. Infatti in questo caso, al posto dei sensori e controllori del BGE si usano blocchi di When, che causeranno reazioni definite dai blocchi di Do ad essi connessi, con questi ultimi che fanno le veci degli attuatori del BGE. Si nasconde perciò il funzionamento dei mattoni logici del BGE e sarà compito del lavoro di tesi tradurre questa logica più semplice e intuitiva in quella originaria, quando si dovrà riportare il lavoro dell'utente in Blender.

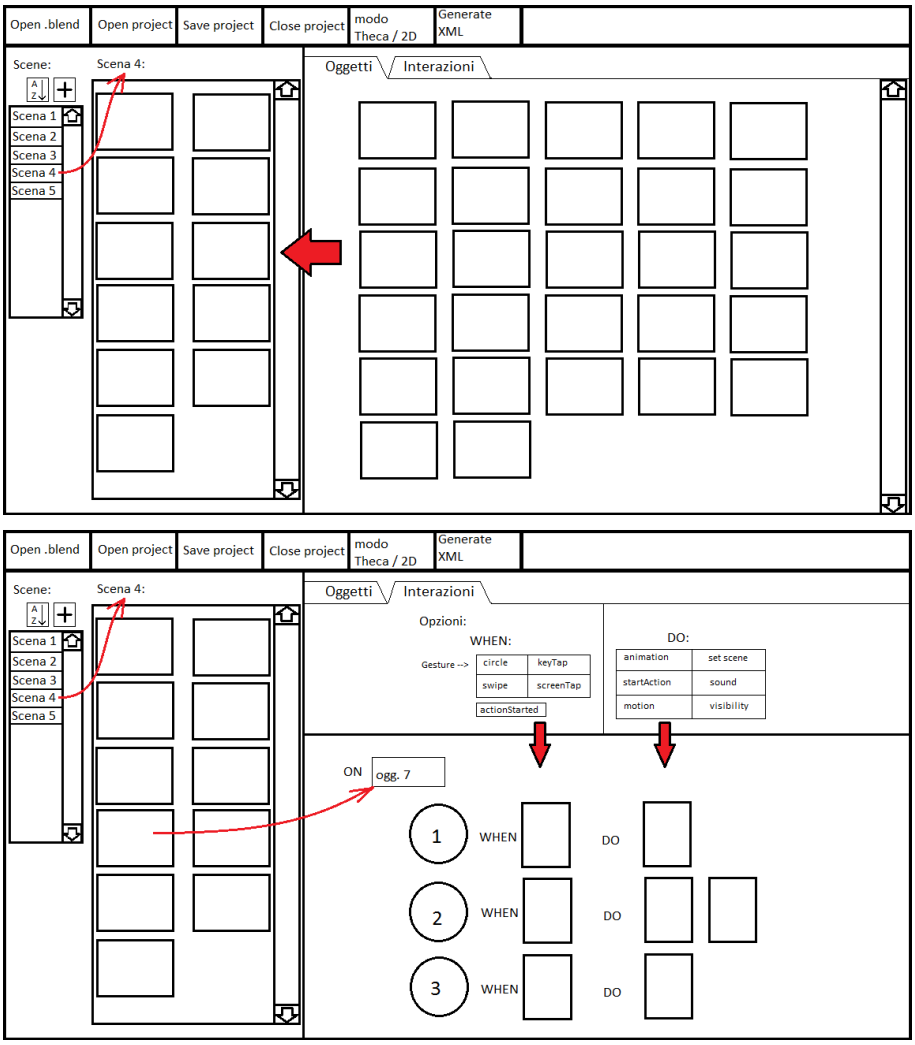


Figura 44. Uno dei primi abbozzi di interfaccia.

Una delle prime interfacce stilizzate è stata quella visibile in Figura 44, in cui, prendendo spunto da Scratch, si dedica uno spazio fisso per visualizzare i blocchi logici disponibili, da trascinare nell'area editor dell'oggetto corrente. Il mockup in questione è stato presto scartato, in quanto si è compreso subito che, per le necessità del lavoro, non è conveniente avere una lista dei blocchi logici da trascinare.

Ad esempio, per il fatto che, a differenza di Scratch, i blocchi richiesti necessitano di essere successivamente definiti internamente: se si desiderasse un blocco animazione, si dovrebbe anche poi indicare il nome dell'animazione e i frame da eseguire, troppi dati da poter racchiudere dentro un blocco di piccole dimensioni. Dunque, la soluzione scelta è che quando si vuole definire un blocco si apra una finestra apposita per tale compito. Inoltre, la soluzione del mockup considerato non andrebbe bene perché mostrerebbe il contenuto di un solo oggetto alla volta, mentre si desidererebbe poter definire le interazioni tra oggetti diversi in modo veloce e avere un quadro più generale della situazione, in ogni istante.

Rispetto a Kodu e Project Spark, si vuole poter definire delle interazioni tra oggetti diversi in modo quanto più intuitivo possibile. Nel Blender Game Engine (dove poi si dovrà riportare il tutto), così come nel Leap Embedder, ciò si traduceva nel creare messaggi inviati da un oggetto, che venivano ricevuti dall'altro oggetto in ascolto. Per cercare di nascondere all'utente il concetto di messaggio e sostituirlo con qualcosa di più semplice, si è scelta la soluzione più veloce ed immediata possibile: tirare una linea che colleghi i due punti. In questo modo, se si vuole che un'azione sull'oggetto "A" scateni una reazione sull'oggetto "B", basterà tirare una linea che unisca l'azione di "A" alla reazione di "B".

Ciò chiaramente implica che si dovrà avere ben visibili tutti gli oggetti di una scena con le loro logiche, per poter così interagire con quelli che si desiderano in ogni momento.

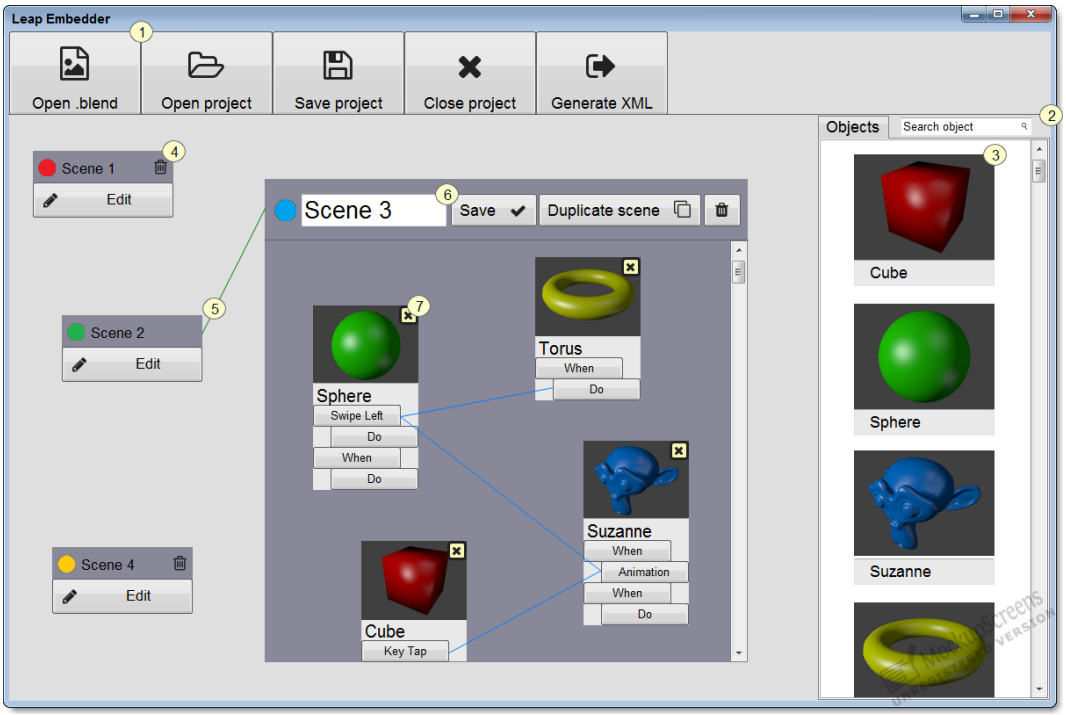


Figura 45. Mockup dell'interfaccia nella sua versione finale.

Utilizzando un programma ad hoc, si sono realizzati alcuni ulteriori mockup dell'interfaccia, delle anteprime attraverso le quali farsi un'idea di come potessero risultare le impostazioni immaginate.
In Figura 45 è mostrato un mockup dell'interfaccia finale. Presenta in alto i comandi di gestione del file (1) e sulla destra una colonna di tutti gli oggetti esistenti nel file Blender su cui si lavora, con un campo di ricerca per filtrare su essi (2) e con ogni oggetto rappresentato graficamente da un'immagine (3). Nel pannello centrale sono visualizzate in formato ridotto tutte le scene create (4) e delle linee indicano eventuali collegamenti presenti tra di esse (5). Una sola scena alla volta può essere espansa (6) per essere modificata, mostrando gli oggetti che possiede al suo interno e come questi interagiscano tra di loro (7).
Per rappresentare la logica di un singolo oggetto interno a una scena, la struttura When-Do viene mantenuta quanto più compatta possibile, siccome si dovrà visualizzarla per molti oggetti contemporaneamente. I When e i Do si differenziano per una rientranza dei secondi, che facilita il colpo d'occhio, un po' come visto in Scratch o nelle righe annidate di Kodu e Project Spark, righe che in questo caso vengono mostrate incolonnate.
In Scratch si è trovato anche un altro concetto utile da recuperare per il nuovo progetto, ovvero sfruttare i colori per facilitare un veloce riconoscimento. In Scratch colori diversi permettono di distinguere le differenti categorie di blocchi logici. Nell'applicazione realizzata, i blocchi logici sono già ben separati esteticamente in blocchi When e Do e non era così fondamentale ricorrere a un ulteriore aiuto cromatico; al contrario, se si devono creare molte scene e altrettante interazioni tra di esse, ci si può ritrovare ad avere su schermo molte linee di collegamento tra le scene, che possono sovrapporsi fino a rendere intricato un rapido riconoscimento del loro percorso. In questo senso, si è scelto di assegnare un colore diverso e personalizzabile a ogni scena, così le linee indicanti interazioni tra scene avranno il colore della scena da cui partono.

4.1.3 Analisi interfaccia e funzionamento

Predisposto l'aspetto generale e il funzionamento desiderato del programma, si è proceduto a realizzare l'interfaccia grafica vera e propria, che ha subito piccoli accorgimenti e modifiche, come l'aggiunta di funzionalità secondarie atte a migliorarne la fruibilità.
Una volta avviato tramite la sua icona, il programma Visual Scene Editor si presenta vuoto come in Figura 46, in attesa che l'utente proceda a indicare il file Blender su cui desidera lavorare.

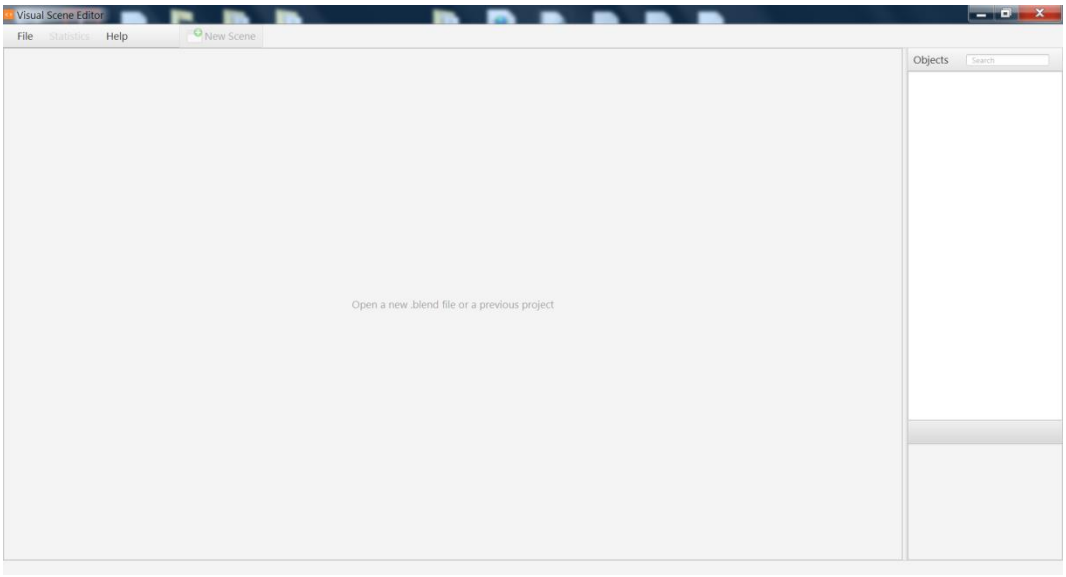


Figura 46. Schermata iniziale del programma.

Al centro, una scritta suggerisce di procedere ad aprire un file di Blender (in formato ".blend") oppure un precedente progetto. Perciò il primo passo da eseguire è quello di aprire il menu File in alto a sinistra e selezionare l'azione desiderata.

4.1.3.1 Menu File

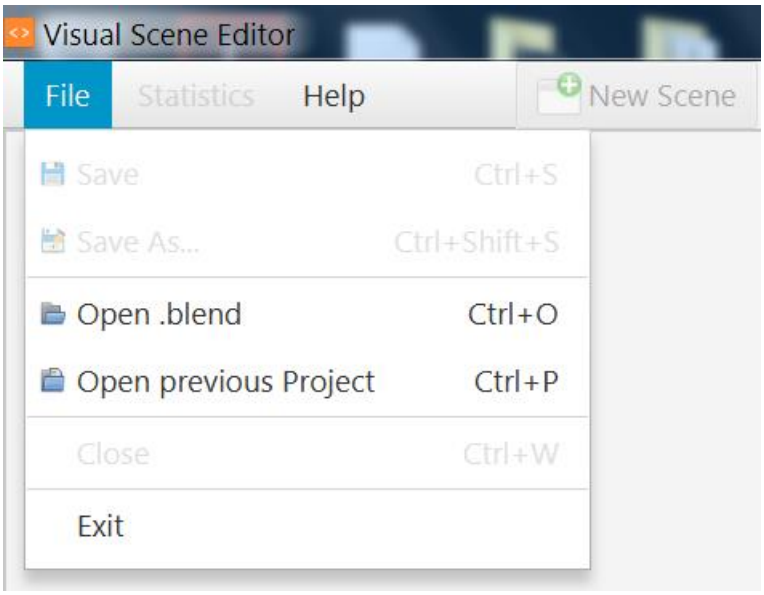


Figura 47. Menu File all'apertura del programma.

Aprendo il menu, le opzioni messe a disposizione sono quelle visibili in Figura 47, descritte di seguito.

- Save: salva il progetto in corso sotto forma di file XML. Se è la prima volta che si salva richiede di salvare assegnando un nome al progetto.
- Save As…: salva con nome il progetto il corso.
- Open .blend: Apre un file Blender con cui iniziare un nuovo progetto.
- Open previous Project: apre un progetto esistente (.xml), precedentemente salvato.
- Close: chiude il progetto corrente.
- Exit: chiude l'applicazione.

Se si salva il progetto o si apre un progetto esistente, il nome del file XML corrente resterà visibile nella barra del titolo del programma in alto. Come si può notare sempre in Figura 47, ogni azione ha una combinazione di tasti rapidi da tastiera con cui poter essere richiamata velocemente, mentre all'avvio del programma i comandi Save e Close sono disabilitati non essendoci ancora un progetto aperto da gestire.
Dunque per iniziare un nuovo progetto si sceglierà di aprire un file Blender. Una volta selezionato il file desiderato, il programma inizia a leggerne tutto il contenuto e al termine visualizza una schermata come quella in Figura 48.

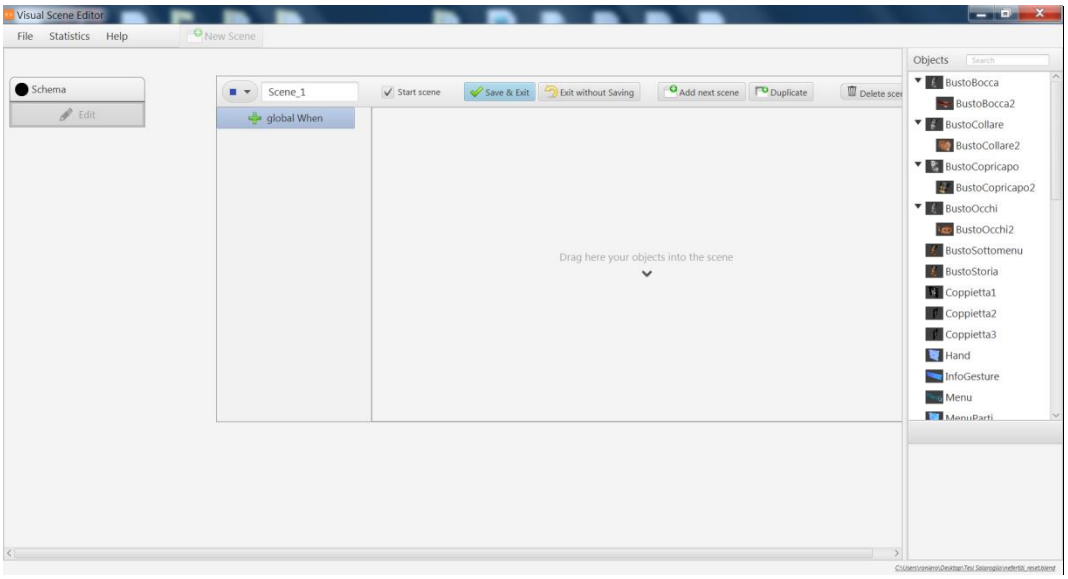


Figura 48. Esempio di schermata visualizzata dopo l'apertura di un file .blend.

4.1.3.2 Colonna degli oggetti

Nella colonna di destra compare una lista gerarchica degli oggetti esistenti in quel file Blender, mentre al centro si apre automaticamente una prima scena con cui poter procedere a definire le logiche di interazione desiderate. Inizialmente la scena sarà vuota e una scritta al suo interno suggerirà di trascinarvi all'interno gli oggetti che si desidera siano presenti in essa. Invece, il nome del file Blender su cui si sta lavorando resterà sempre visibile nell'angolo in basso a destra, assieme al percorso assoluto del file.

Figura 49. Colonna degli oggetti.

Prima di continuare la descrizione della procedura da seguire per realizzare le scene desiderate, ci si sofferma ad analizzare nel dettaglio la colonna degli oggetti a destra, visibile in Figura 49. Questa si compone di un pannello contenente i nomi di tutti gli oggetti del file Blender (2). Ciascun oggetto è rappresentato da una piccola immagine che aiuta il riconoscimento veloce e dal suo nome. La visualizzazione gerarchica ad albero riprende quella del programma Blender, con un oggetto padre che può contente uno o più oggetti figli; in tale caso il padre può essere ridotto per nascondere i suoi figli. Un campo di ricerca in alto (1) permette di filtrare la lista di oggetti in base al nome desiderato: la lista filtrata si aggiornerà in tempo reale mentre l'utente digita nel campo di ricerca e se un oggetto corrispondente alla ricerca possiede un padre, quest'ultimo resterà visualizzato per segnalare all'utente

tale gerarchia. Se si seleziona un oggetto (3) esso verrà visualizzato nell'area sottostante (4): in questa sezione viene mostrata l'immagine che lo rappresenta in un formato più grande, così per ogni oggetto con cui l'utente interagisce si avrà sempre una visualizzazione più chiara di quale esso sia.
Gli oggetti in questo pannello includono anche eventuali luci presenti, ma non comprendono le camere, in quanto queste ultime non devono essere gestite dall'utente (generando in Blender le scene create nel Visual Scene Editor, le camere verranno generate come necessario). Invece, le luci sono a discrezione dell'utente, che può decidere se e in quali scene inserirle.
Affinché il programma visualizzi un'immagine rappresentativa di ogni oggetto del file, l'utente dovrebbe prima di tutto avviare in Blender uno script Python "rendOnFiles.py" appositamente realizzato, che è richiamabile nella 3D View sia da barra spaziatrice sia attraverso un pulsante "Render all objects on files" aggiunto nel menu Object. Questo add-on, una volta invocato, genera una cartella nella stessa directory in cui si trova il file Blender ed all'interno salva un'immagine renderizzata per ogni oggetto, con le opportune illuminazioni, rotazioni e inquadrature. In questo modo, il programma Visual Scene Editor, rilevando l'esistenza di tale cartella nella stessa directory in cui ha fatto accesso al file Blender, può mostrare queste immagini. Per le luci, dato che non è possibile generarne un rendering, si ricorre a un'immagine di default che le distingua.

4.1.3.3 Le scene

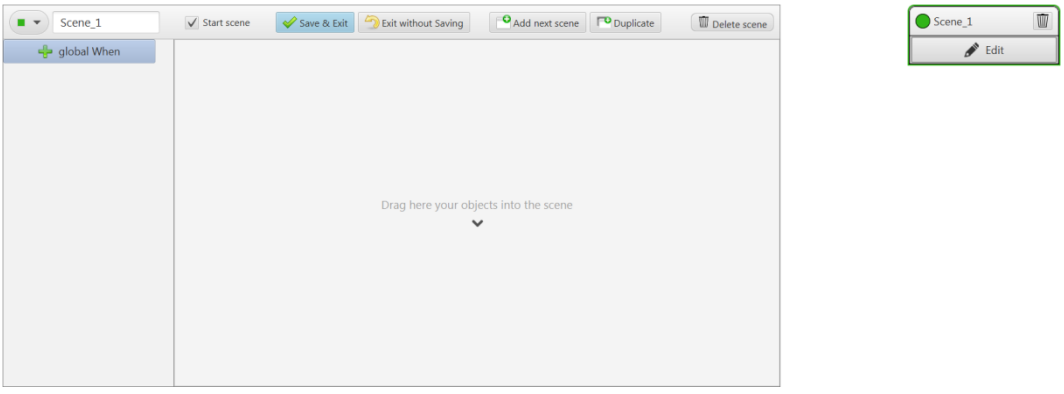
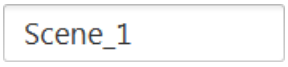


Figura 50. Rappresentazione grafica di una stessa scena vuota in versione aperta (a sinistra) e chiusa (a destra).

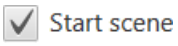
Come detto in precedenza, all'apertura di un file Blender il programma mostrerà automaticamente una prima scena aperta da cui partire nella definizione del proprio progetto. Una scena espansa occupa buona parte del pannello centrale del programma e nella parte superiore presenta una toolbar con i vari comandi per la sua gestione (Fig.50). Se ci si posiziona su ciascuno dei comandi illustrati, un tooltip

appare per fornirne una spiegazione. Da sinistra verso destra, sono visualizzati i controlli descritti nel seguito.

Permette di selezionare un colore da assegnare alla scena, scegliendolo attraverso un completo selettore del colore.

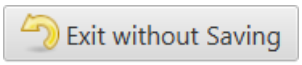


Campo testuale in cui è possibile definire il nome della scena. Se si crea una scena nuova, questa possiederà inizialmente un nome corrispondente a "Scene_" seguito dal numero della scena ennesima. Non è permesso assegnare a una scena un nome già esistente o includere i caratteri '.' ':' e ' ', questo sia per evitare problemi di gestione sia per questioni di compatibilità con i nomi assegnabili alle scene dentro il programma Blender.

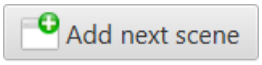


Questa checkbox indica se la scena deve essere considerata o meno come quella di partenza da cui avviare l'intera simulazione. Naturalmente una sola scena potrà essere settata come scena di start, se si prova a spuntare questo campo quando ne esiste già una verrà chiesto se si desidera sostituire quella precedentemente indicata. La prima scena di un nuovo progetto sarà sempre spuntata quale scena di partenza di default.

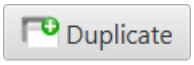
Pulsante che salva le modifiche apportate alla scena aperta e la chiude. Il colore azzurro lo fa risaltare, essendo anche il pulsante associato al comando rapido della pressione del tasto Invio da tastiera.



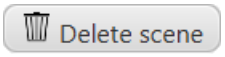
Pulsante che chiude la scena senza salvare eventuali modifiche apportate.



Chiude questa scena e ne apre una nuova. Se rileva modifiche non salvate chiede se si desidera salvare prima di chiudere questa scena. La nuova scena verrà posizionata verticalmente sotto di questa nel pannello generale.



Duplica la scena, aprendone una nuova che possiederà al suo interno gli stessi oggetti di questa e le stesse interazioni al suo interno e verso altre scene. Se rileva modifiche non salvate chiede se si desidera salvare prima di chiudere questa scena. La nuova scena verrà posizionata verticalmente sotto di questa nel pannello generale.



Elimina la scena attuale, cancellando quindi anche tutte le eventuali interazioni che aveva con le altre. Una scena eliminata non sarà più recuperabile.

Sotto la toolbar illustrata, la scena aperta si compone di un pannello contenente gli oggetti interni a essa e di una colonna dei "global When" sulla sinistra, che si illustrerà più avanti.
Una scena chiusa viene rappresentata in modo simile a come in seguito si vedrà avviene per un oggetto chiuso. La visualizzazione è limitata al nome, un'icona a cestino per eliminarla e un pulsante di Edit per aprirla e modificarne le caratteristiche. In aggiunta, una scena chiusa presenta un cerchio colorato accanto al nome che ne indica il colore associato e la scena che è stata dichiarata quale quella di partenza della simulazione si distinguerà dalle altre risaltando grazie ad un bordo colorato del suo colore. In caso ci sia una scena aperta, tutte le altre scene chiuse presenti nel pannello generale verranno momentaneamente spostate, in modo da disporsi attorno a quella aperta e non venire coperte da essa, per poi tornare alle loro posizioni originali quando verrà chiusa. Inoltre verranno disabilitati i pulsanti di Edit e di rimozione scena a tutte le altre scene, fintanto che non si chiuderà la scena aperta. Se si vuole aggiungere una nuova scena, si deve selezionare il pulsante "New Scene" posizionato nella barra superiore al pannello generale, che genera una nuova scena chiusa collocandola affiancata a destra dell'ultima scena chiusa esistente. Anche questo pulsante risulta disabilitato quando si ha una scena aperta.

4.1.3.4 Gli oggetti

Per aggiungere un oggetto a una scena l'utente deve trascinarlo all'interno della scena aperta, prelevandolo dalla colonna degli oggetti a destra, con un'operazione di drag-and-drop. Non è permesso aggiungere più volte un oggetto alla stessa scena, mentre se si aggiunge un oggetto che possiede altri oggetti figli, questi saranno automaticamente aggiunti assieme al padre nella scena. L'oggetto verrà inserito spazialmente nella scena nell'esatto punto in cui il drag viene terminato, ma l'utente potrà sempre successivamente spostarlo all'interno della scena aperta trascinandolo dove preferisce. Ogni oggetto aggiunto a una scena viene inizialmente rappresentato nella visualizzazione di oggetto espanso, come in Figura 51. Presenta una toolbar superiore indicante il suo nome, con a destra un'icona di un cestino da dove poter rimuovere l'oggetto dalla scena. Subito sotto è presente l'immagine che rappresenta l'oggetto, con tre pulsanti a destra di essa. I primi due sono dei toggle button che possono essere abilitati o meno per indicare la visibilità dell'oggetto nella scena e se tale oggetto debba fungere da puntatore in essa, mentre il terzo è un pulsante di Edit che permette di ridurre la visualizzazione

dell'oggetto in un formato minimo. Esso può risultare utile quando ci si ritrova a dover gestire molti oggetti in una scena oppure quando non si ha interesse di osservare i dettagli interni di questo oggetto, pertanto né si riduce lo spazio occupato nascondendone i dettagli, inclusa la logica di interazione, in modo simile a come si fa per aprire e chiudere una scena.

Il concetto di oggetto visibile o meno in una scena è lo stesso di Blender e non richiede quindi specifiche spiegazioni, mentre la funzione di puntatore necessita di chiarimenti. Se si dichiara un oggetto quale puntatore nella scena, esso si comporterà come un puntatore del mouse. Questo significa che se il progetto verrà alla fine eseguito sul monitor di un computer, l'oggetto puntatore sostituirà esteticamente a schermo il classico puntatore del mouse. Se invece si vorrà eseguire il progetto nella teca olografica, sarà fondamentale avere un oggetto puntatore che si sposti nello spazio seguendo i movimenti della mano, siccome nella teca non è presente alcun puntatore del mouse. Altrimenti, sarebbe impossibile capire dove si stia puntando con la mano, rendendo praticamente impossibile eseguire i gesti sugli oggetti desiderati. Pertanto, conviene sempre che l'utente definisca un oggetto puntatore per ogni scena e un messaggio di allerta impedisce che l'utente possa impostare più di un puntatore nella stessa scena, chiedendo se si desidera sostituire il puntatore precedente.

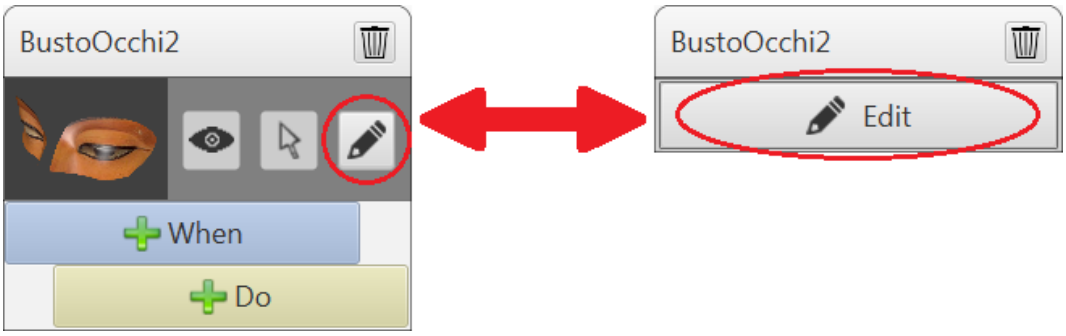


Figura 51. Passaggio tra rappresentazione oggetto espanso e ridotto.

Infine, la parte inferiore di ogni oggetto è composta da una lista dei costrutti logici di interazione associati all'oggetto. Come descritto in precedenza a proposito dei mockup, si è scelto di ricorrere a pulsanti di When e di Do indicanti rispettivamente le condizioni scatenanti e le azioni che ne conseguono. Inizialmente, ogni oggetto dispone di un When e un Do vuoti. I When si distinguono immediatamente dai Do grazie a notazioni secondarie come un differente colore e uno stile d'indentazione rappresentato dal fatto che i Do risultano scalati verso destra, per suggerire che sono azioni scatenate dal When sopra di essi. Se si procede ad impostare uno dei due pulsanti iniziali, l'oggetto si espanderà verticalmente verso il basso per visualizzare una nuova coppia di When-Do vuota a disposizione dell'utente. Inoltre si vuole permettere di poter dichiarare più di un Do scatenato da un'unica condizione When; perciò, quando si setta un pulsante di Do è garantito che ce ne sia sempre un

altro vuoto a disposizione per quel When. In Figura 52 sono mostrati alcuni esempi di come si possa espandere verticalmente un oggetto in base a quali pulsanti della logica vengono dichiarati.

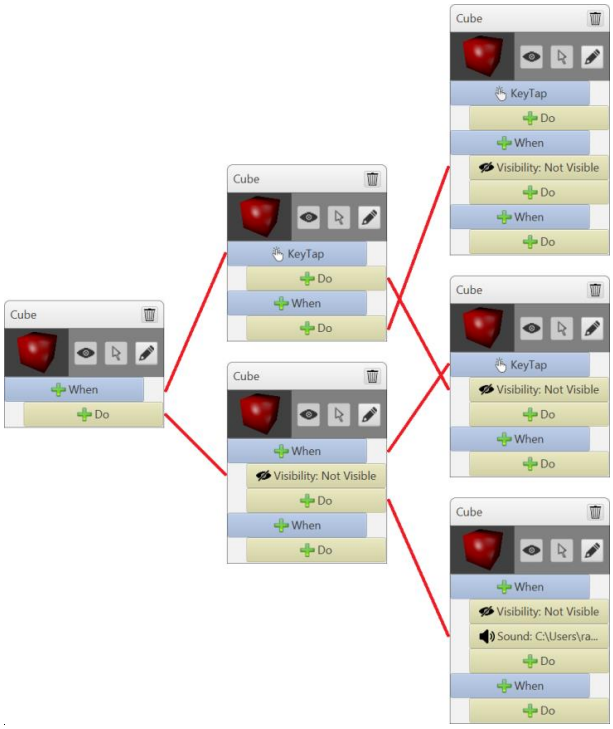


Figura 52. Da sinistra verso destra, esempi di progressione nella definizione di When e Do.

Per le definizioni dei When e dei Do si rimanda alla sezione apposita, dove sono illustrati tutti i possibili blocchi logici dichiarabili dall'utente. Una volta definito un When o Do, esso indicherà il nome del comando che è stato selezionato, assieme ad una relativa icona identificativa. Se questa descrizione non fosse completamente visibile nello spazio disponibile, posizionandosi su di esso appare un tooltip che mostra l'indicazione completa.

4.1.3.5 Le linee di collegamento

Come si è descritto, se si desidera che un evento che avviene su di un oggetto scateni un'azione su di esso è sufficiente definire una coppia di When e Do interna a esso. Invece, nel caso si volesse che un evento su un oggetto scatenasse una reazione in un oggetto differente (o più di uno) si deve ricorrere ai link. Si ipotizza per esempio che si desideri permettere che selezionando un oggetto 'A' si avvii un'animazione dell'oggetto 'B': in questo caso si definirà un When apposito in 'A' (senza definirne Do) e in 'B' si dichiarerà il Do desiderato (senza definire il When sopra di esso); infine per indicare che questi due blocchi logici sono connessi tra di loro da una relazione di causa-effetto, si tira a mano (con

un drag-and-drop) una linea che partendo dal When di ‘A’ si connette al Do di ‘B’. Questo link rappresenterà esteticamente un collegamento creato tra i due blocchi.
Un link può sempre solo essere tirato a partire da un When di un oggetto o When globale della scena, mentre terminerà sempre su un Do, a eccezione di quando si vuole realizzare un cambio scena. Infatti per indicare un passaggio di scena si deve tirare in modo analogo una linea da un When alla toolbar della scena di arrivo e, rilasciandola su di essa, la linea si aggancerà alla scena: se si vuole fare un cambio di scena si rilascerà su una scena chiusa, mentre se si desidera ricaricare la scena attuale sarà possibile farlo rilasciando il link sulla parte sinistra della toolbar della scena aperta. Se si termina il trascinamento della linea in un qualsiasi altro punto non valido, essa non sarà agganciata con successo e sparirà. Durante il trascinamento, se si passa su una zona dove può essere collegato verrà segnalato mostrando il link che anziché seguire il puntatore si aggancia momentaneamente all’elemento presente in quel punto. Tutte le linee tirate escono dal lato destro di un When per entrare dal lato sinistro di un Do o scena, così si semplifica la distinzione tra l’estremità di partenza e di arrivo. Inoltre i link assumono sempre il colore della scena da cui partono. Questo permette di facilitare il riconoscimento delle connessioni, soprattutto quando ci si ritrova a doverne gestirne molte su schermo che possono finire per intrecciarsi. L’utente può scegliere se tirare la linea di collegamento prima o dopo aver definito i pulsanti di When e Do alle sue estremità. Non è richiesto un ordine nel realizzare le operazioni e, volendo, si può chiudere la scena definendo solo il Do per poi tornare a riaprirla e dichiarare anche il relativo When, ma in ogni caso se si termina il progetto con un When vuoto eventuali Do conseguenti a esso non verranno mai eseguiti, a meno che non siano il punto di arrivo di link provenienti da altri oggetti. Allo stesso modo un link verrà scartato in fase di importazione del progetto su Blender se le sue estremità non sono state entrambe dichiarate.
In Figura 53 è mostrato un esempio di progetto avente una scena aperta, con le linee di collegamento che rappresentano le interazioni tra gli oggetti della scena o tra le differenti scene.

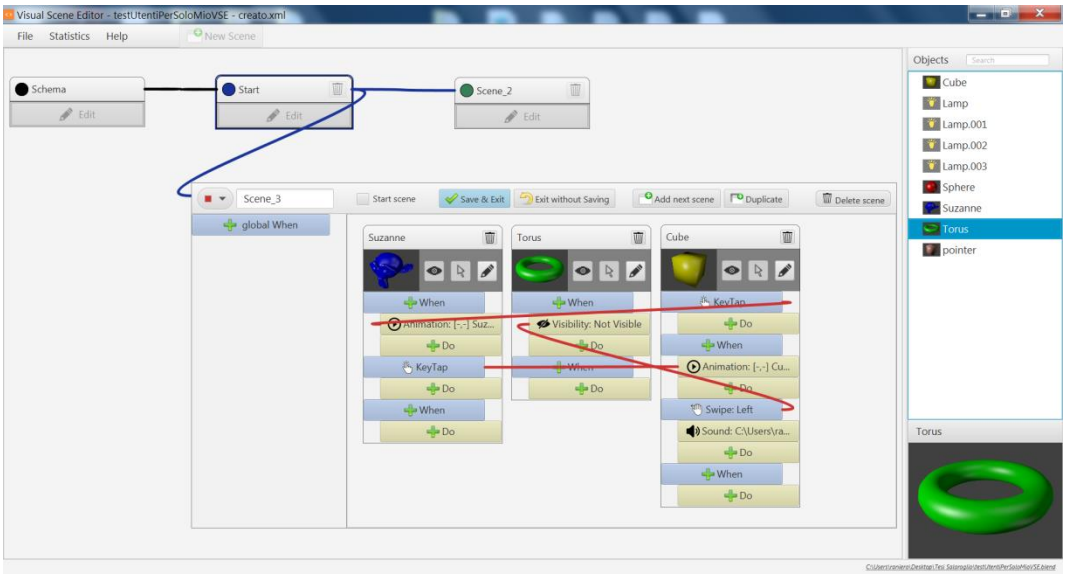


Figura 53. Un esempio di progetto in realizzazione, con una scena aperta, i suoi oggetti interni e le varie logiche di interazione.

Nel caso si riduca un oggetto attraverso il pulsante di Edit, i suoi When e Do non risulteranno più visualizzati, perciò tutte le linee in entrata e in uscita da essi verranno agganciate alla toolbar superiore dell'oggetto.

Se si chiude una scena, riportandola alla visualizzazione minima, si nasconderanno gli oggetti contenuti in essa e le eventuali linee di interazione che li collegavano, ma se da uno dei vari When si scatenava un cambio di scena, con un collegamento a un'altra scena esterna, si mostrerà questa informazione rappresentandola con un link che parte dall'estremità destra della toolbar della scena che si ha chiuso verso l'estremità sinistra della toolbar dell'altra scena (che può risultare chiusa o successivamente aperta).

Se anziché creare blocchi When e Do si desiderasse eliminarli, si deve cliccare su di essi con il tasto destro per far comparire un menu contestuale (Fig.54). Da questo menu si può decidere di eliminare il blocco corrente oppure rimuovere una particolare linea che partiva o terminava in esso.

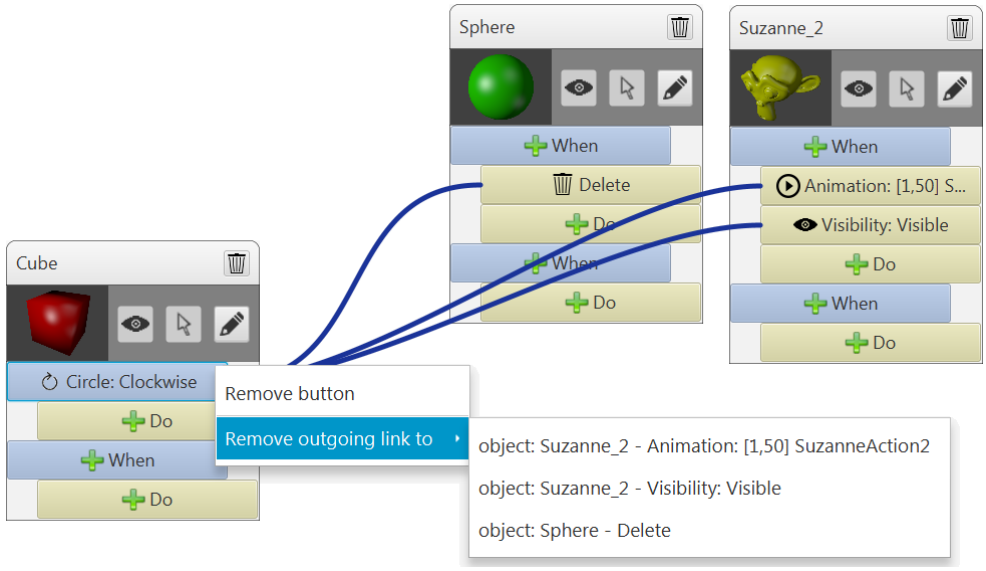


Figura 54. Menu contestuale per rimozione della logica.

4.1.3.6 I When globali

Come si era mostrato, una scena aperta possiede una colonna di sinistra che include i "global When". Questi cosiddetti When globali si gestiscono come dei normali When, ma rappresentano quelle condizioni che si desidera vengano sempre rilevate dentro la scena, in ogni punto. Per esempio si può desiderare che avvenga un cambio di scena quando viene eseguito un particolare gesto in un qualsiasi punto, anziché specificarlo solo su un oggetto, così si dichiara un When opportuno in questa colonna e da esso si tirerà una linea diretta all'altra scena a cui passare. Si segnala che questi When globali, non possedendo dei Do sotto di essi, affinché producano un effetto richiedono una linea che punti a una scena, per il cambio scena, o a uno o più Do di oggetti interni a questa scena, per azioni su di essi.

4.1.3.7 Scena Schema

Si era potuto notare in Figura 48 che, fin dall'avvio di un nuovo progetto, si dispone anche di una scena di nome Schema posizionata nell'angolo in alto a sinistra del pannello generale delle scene. Questa scena è diversa dalle altre e ha uno scopo specifico. La scena Schema non corrisponde a una reale scena, ma rappresenta un comportamento generalizzato per tutte le scene della simulazione, una sorta di "template": ciò che si definisce al suo interno verrà considerato applicato a ogni scena.

Esteticamente è rappresentata come una normale scena, con la differenza che non può essere rinominata o eliminata e nel pannello generale non è possibile spostarla, ma resta ancorata nell'angolo superiore. Inoltre, non essendo una classica scena, quando è aperta non possiede gli usuali pulsanti per duplicarla, aggiungere una scena successiva a essa oppure settarla quale scena di start.
Per spiegare meglio l'utilità della scena Schema, può capitare spesso di avere degli oggetti che si desidera siano presenti in ogni scena della simulazione: in questo caso basterà inserirli una sola volta in questa scena anziché ripetere l'operazione per ogni singola scena che si crea e il programma provvederà automaticamente a inserirli in tutte le scene quando si importerà il risultato in Blender. Di solito alcuni degli oggetti che capita frequentemente di voler ripetere in ogni scena sono le luci e l'oggetto puntatore. Se per esempio si desidera che tutte le scene abbiano un certo oggetto che reagisca in uno stesso modo a un particolare gesto ad eccezione di una sola scena in cui non rispetta tale comportamento ma scatena un'azione diversa, lo si può inserire nella scena Schema definendone la logica comportamentale diffusa e la sola scena differente dalle altre può ridichiarare quest'oggetto con un nuovo comportamento esclusivo di tale scena che varrà come eccezione locale rispetto alla regola generale indicata. Questo permette di fare in modo che in quella scena prevalga la logica specifica della scena stessa anziché quella di Schema. Infatti il programma nelle situazioni di conflitto tra la scena Schema e una scena particolare farà sempre prevalere quest'ultima più specifica rispetto a quella generica, nello stesso modo in cui un When interno a un oggetto prevale su un'uguale tipologia di "global When" della scena. Una tale gestione di gerarchie di priorità permette di evitare che l'utente possa definire situazioni ambigue.

4.1.3.8 La barra di stato inferiore

Infine si segnala la presenza di una barra di stato nella parte inferiore del programma (Fig.55). In questa barra viene sempre visualizzato, nell'estremità destra, il percorso completo del file Blender su cui si sta lavorando nel progetto in corso. Il lato sinistro della barra è riservato a eventuali messaggi di warning da visualizzare che si riferiscono non a un vero e proprio problema (per quelli esistono finestre apposite), ma a situazioni che potrebbero nascondere una situazione indesiderata non gestita. Un esempio è quando si chiude una scena senza aver inserito al suo interno alcun oggetto.

Figura 55. Barra di stato inferiore, con warning di scena vuota a sinistra e file Blender a destra.

4.1.3.9 Statistiche

La barra dei menu superiore del programma, oltre al primo menu File analizzato inizialmente, possiede anche altri due voci: una per visualizzare delle statistiche del progetto aperto ed una per informazioni generali sul programma Visual Scene Editor. Selezionando “Statistics” si aprirà una finestra in cui viene visualizzato un diagramma a barre orizzontali sovrapposte, che riassume le scene create nel progetto attualmente in corso, con i loro oggetti (Fig.56). Sull’asse delle ordinate sono indicate tutte le scene, mentre su quello delle ascisse si conteggiano gli oggetti presenti dentro di esse. La scena Schema non viene inclusa tra le scene, non essendo una scena effettiva, ma i suoi oggetti vengono distribuiti dentro tutte le altre e si distinguono anche eventuali oggetti che risultano essere stati dichiarati sia in Schema sia in una scena specifica.

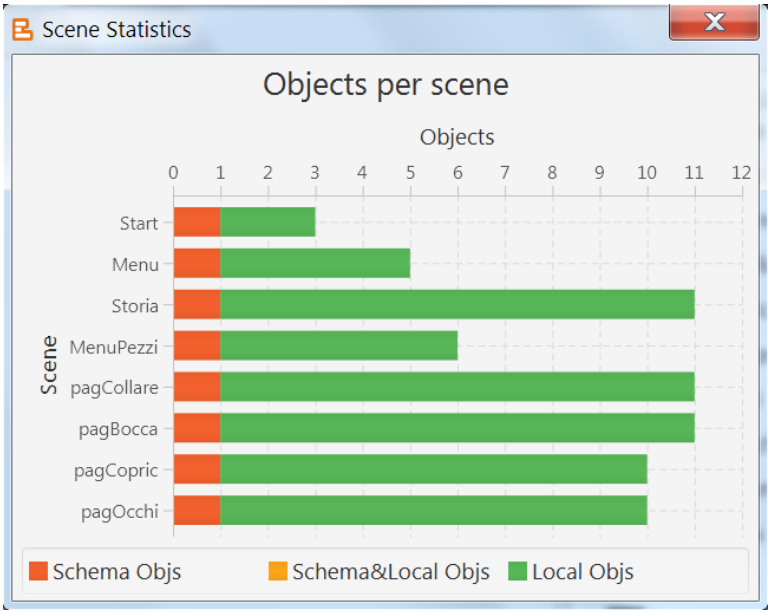


Figura 56. Finestra delle statistiche di un progetto strutturato.

4.1.3.10 When

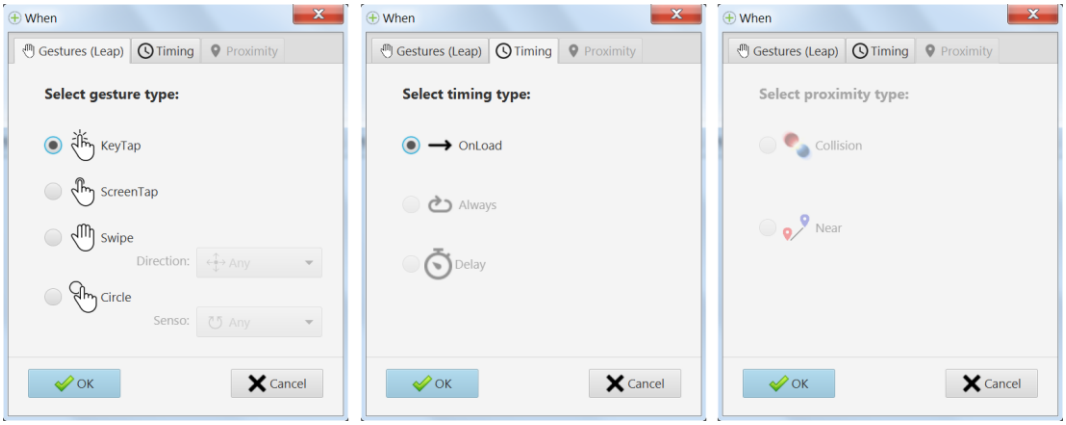


Figura 57. Finestra di selezione di un When, con i suoi tre pannelli disponibili.

Se si desidera impostare un nuovo When o modificarne uno già esistente, occorre aprire la finestra di definizione del When, che distingue le varie tipologie disponibili in più pannelli (Fig.57).

- Gestures (Leap): il primo tab comprende i quattro gesti manuali riconoscibili dal Leap Motion, ovvero KeyTap, ScreenTap, Swipe e Circle.
    - KeyTap (Fig.58)

Figura 58. Dettaglio del selettore KeyTap.

    - ScreenTap (Fig.59)

Figura 59. Dettaglio del selettore ScreenTap.

    - Swipe: un gesto Swipe può essere differenziato in base alla direzione con cui viene eseguito (Fig.60). Si distingue in Swipe Left, Swipe Right, Swipe Up e Swipe Down, mentre se si seleziona lo Swipe Any di default allora questo When si attiverà al rilevamento del tracciamento in una qualsiasi direzione.

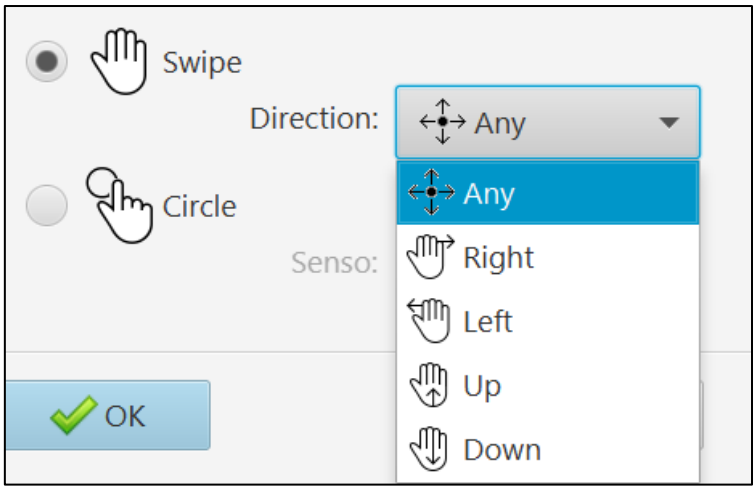


Figura 60. Dettaglio del selettore Swipe.

- Circle: anche in questo caso, un Circle e differenziabile in base al senso con cui viene realizzato (Fig.61). Di default vi è il Circle Any che si attiva indipendentemente dal senso, mentre Circle Clockwise e Circle Counterclockwise distinguono rispettivamente dei gesti circolari orari e antiorari.

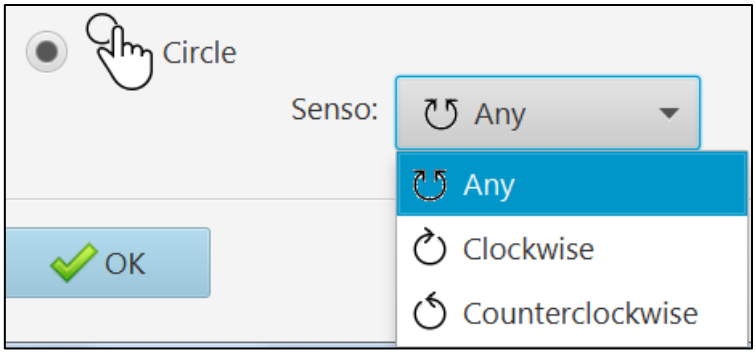


Figura 61. Dettaglio del selettore Circle.

- Timing: il secondo tab riunisce gli eventi temporali. L'unico When attualmente disponibile è il primo di OnLoad, che si rivela utile alle nostre necessità, mentre Always e Delay sono stati lasciati disabilitati in quanto mantenuti come possibili aggiunte e non completamente implementati, dato che per gli scopi prefissati erano meno importanti. Always, come dice il suo nome, corrisponde al sensore Always del BGE, ovvero è sempre attivo. Invece per quanto riguarda Delay, anche questo possiederebbe un omonimo attuatore nel BGE e servirebbe ad attivare un'azione dopo un certo tempo di attesa definito al suo interno. Può essere utile quando si vorrebbe avere due OnLoad che vengano eseguiti in un ordine specifico, allora il secondo si può esprimere con un Delay.

- OnLoad: questo sensore si attiva una sola volta all'inizio (Fig.62). Se si applica come When globale a una scena, permette di eseguire delle azioni al caricamento della scena, ovvero appena si passa a tale scena. Se invece viene applicata a un singolo oggetto, si attiverà alla creazione dello stesso, che di solito coincide anche con l'avvio di quella scena. Di solito, conviene applicarlo all'intera scena quando si vuole attivare varie azioni sugli oggetti interni, mentre si consiglia di applicarlo su un singolo oggetto quando si vuole scatenare un'azione solo su di esso, per mantenere più pulito il pannello. Questo può essere utile per esempio quando si desidera che, appena si passa a una scena, si attivino subito delle animazioni su certi oggetti al suo interno.

Figura 62. Dettaglio del selettore OnLoad.

- Proximity: infine il terzo tab racchiude gli eventi spaziali e risulta disattivato, non essendo completamente implementato. Includerebbe eventi come Collision, che rileva le collisioni tra oggetti, e Near, che si attiva quando un oggetto si avvicina entro un certo limite ad esso.

4.1.3.11 Do

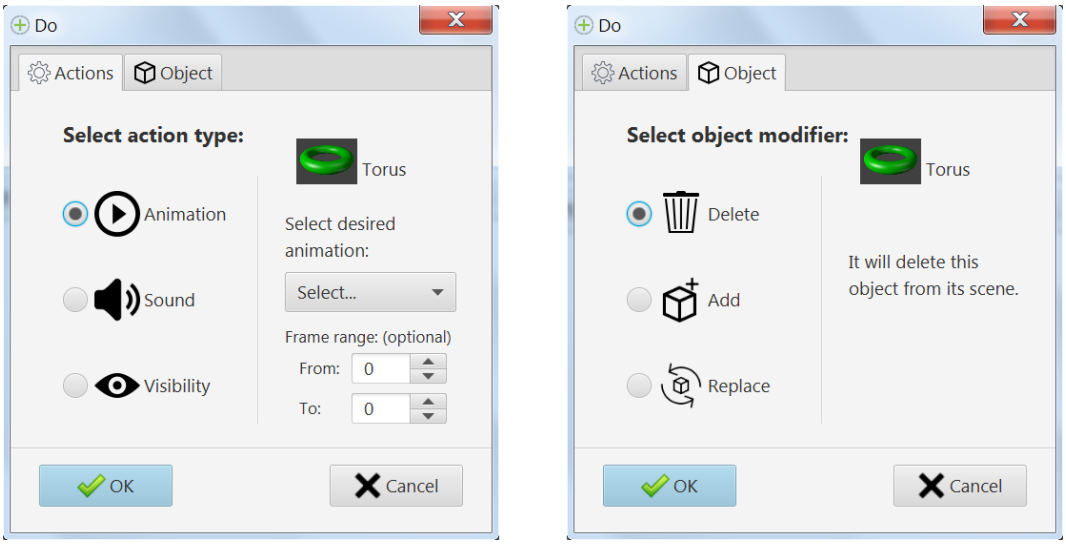


Figura 63. Finestra di selezione di un Do, con i suoi due pannelli disponibili.

La finestra di definizione di un Do visualizzata si compone dei due pannelli mostrati in Figura 63.

- Actions: il primo pannello visualizzato nella finestra di selezione di un Do è quello delle Action, che racchiude le seguenti principali azioni applicabili all'oggetto.
    - Animation: avvia un'animazione dell'oggetto. L'animazione desiderata è selezionabile per nome da una lista comprendente quelle possedute dall'oggetto. Facoltativamente, è permesso selezionare anche solo un intervallo di frame da eseguire, indicando un frame iniziale e uno finale (Fig.64). In questo modo si può scomporre per esempio una singola animazione in due parti distinte, eseguendole separatamente. Se non si specifica i frame di esecuzione, l'animazione verrà eseguita per intero.

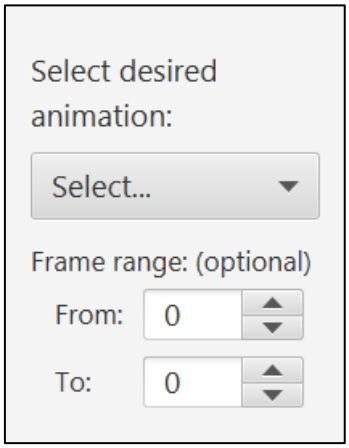


Figura 64. Dettaglio del selettore Animation.

    - Sound: permette di selezionare un qualsiasi file audio presente sul computer da riprodurre (Fig.65). Accetta file audio dei vari formati principali, accettando i formati ".wav", ".mp3", ".aac", ".flac", ".wma".

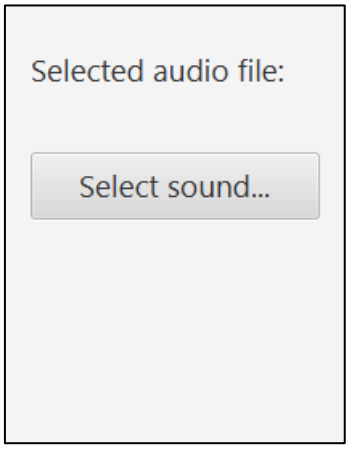


Figura 65. Dettaglio del selettore Sound.

- Visibility: consente di cambiare la visibilità dell'oggetto. Un oggetto può essere settato come "visibile" o "non visibile" (Fig.66). Eventuali oggetti figli di questo oggetto manterranno inalterata la loro visibilità.

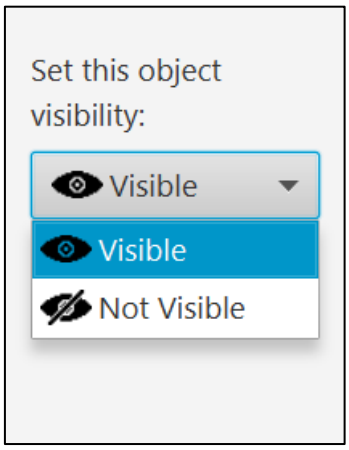


Figura 66. Dettaglio del selettore Visibility.

- Object: un secondo pannello comprende i comandi di gestione di creazione o eliminazione degli oggetti.
  - Delete: come indicato in Figura 67, elimina questo oggetto dalla scena. Un oggetto eliminato non potrà più essere ripristinato uguale a prima nella scena.

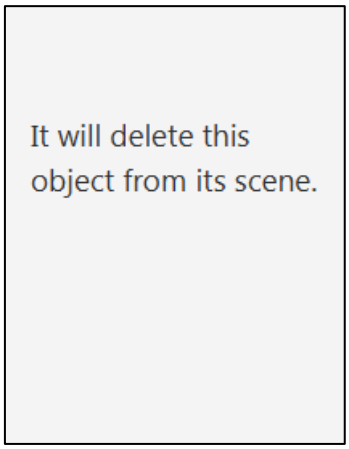


Figura 67. Dettaglio del selettore Delete.

  - Add: aggiunge un nuovo oggetto nel punto in cui si trova quello attuale (Fig.68). L'oggetto aggiungibile è selezionabile da una lista comprendente tutti quelli presenti nel file Blender, escluso l'oggetto corrente.

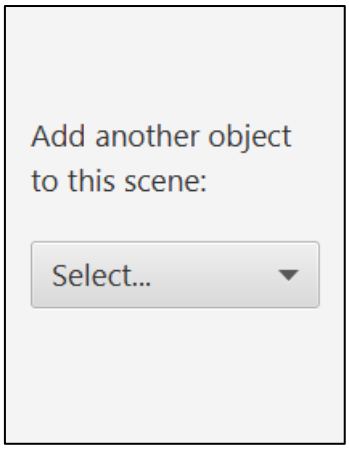


Figura 68. Dettaglio del selettore Add.

- Replace: sostituisce l'oggetto attuale con uno nuovo desiderato (Fig.69). L'oggetto che si vuole inserire è selezionabile da una lista comprendente tutti quelli presenti nel file Blender, escluso l'oggetto corrente che verrà rimosso. Il risultato generato equivarrebbe a quello ottenibile combinando un Delete dell'oggetto attuale con una Add di quello nuovo.

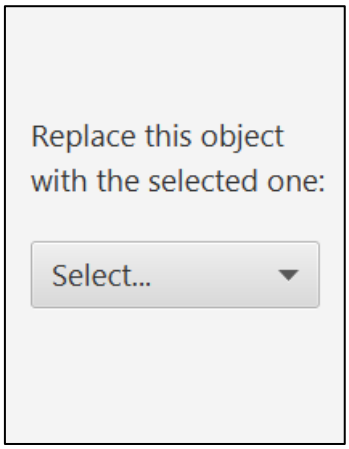


Figura 69. Dettaglio del selettore Replace.

4.1.4 Architettura

Nell'impostare il progetto si è partiti seguendo i buoni principi della progettazione software. Un principio molto importante è quello di seguire il pattern della Model-View-Controller (MVC) [69], illustrato in Figura 70. Un pattern è uno schema di progettazione che aiuta il programmatore

nell'organizzazione strutturale del software e un'architettura software MVC comprende, appunto, i seguenti tre componenti.

- Model: definisce i dati dell'applicazione (di solito i dati sono memorizzati in un database).
- View: fornisce l'interazione che l'utente osserva (come una finestra del programma o una pagina web). Questi componenti offrono i dati all'utente e inviano azioni al Controller per manipolare i dati.
- Controller: essenzialmente, fornisce l'interfaccia tra la View e il Model.

Il pattern architetturale MVC è largamente utilizzato per lo sviluppo di interfacce grafiche utente. Uno dei benefici dell'architettura software MVC è l'astrazione dei tre componenti ad alto livello. Un Model non deve avere conoscenza della View che è fornita all'utente.

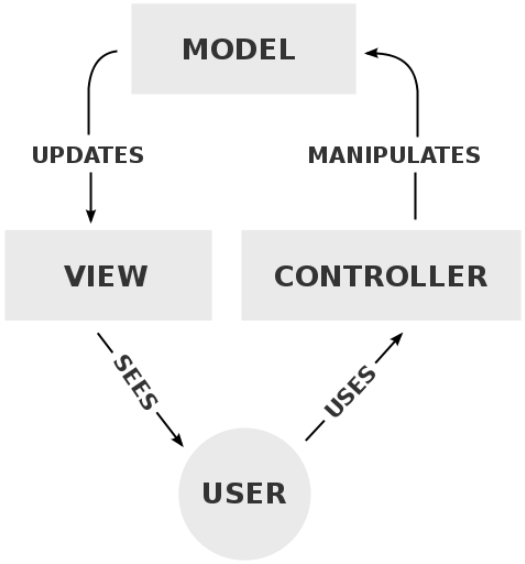


Figura 70. Schema Model-View-Controller. (fonte: wikipedia.org)

La modellazione del software si è basata sul modello di pattern MVC. Seguendo esso, si è suddiviso il codice in varie unità, creando un package per ognuna di queste, come riportato di seguito.

- application: contiene la classe principale "MainApp" (Fig.71), che viene eseguita all'avvio del programma. Questa come prima azione provvede, attraverso il metodo *start()*, a settare graficamente l'applicazione, ricorrendo a *initSceneOverview()* per inizializzare il pannello principale e *showMenuBar()* per la barra dei menu superiore. Si segnala il metodo *loadBlendDataFromFile()*, che viene chiamato dalla classe "MenuLayoutController" quando l'utente sceglie di iniziare un nuovo progetto e pertanto il metodo legge un nuovo file Blender,

ricavandone tutte le informazioni necessarie contenute al suo interno e provvede a generare una lista degli oggetti presenti. Anche i metodi *saveProjectDataToFile()* e *loadProjectDataFromFile()* fanno riferimento ai comandi del menu per il salvataggio di un progetto e la riapertura di un precedente progetto. Invece i metodi *showWhenSelectionDialog()* e *showDoSelectionDialog()* si occupano rispettivamente di aprire una finestra di selezione di un nuovo When e Do, relegandone la gestione alle rispettive classi controllore; lo stesso vale per il metodo showStatistics() per la finestra delle statistiche.

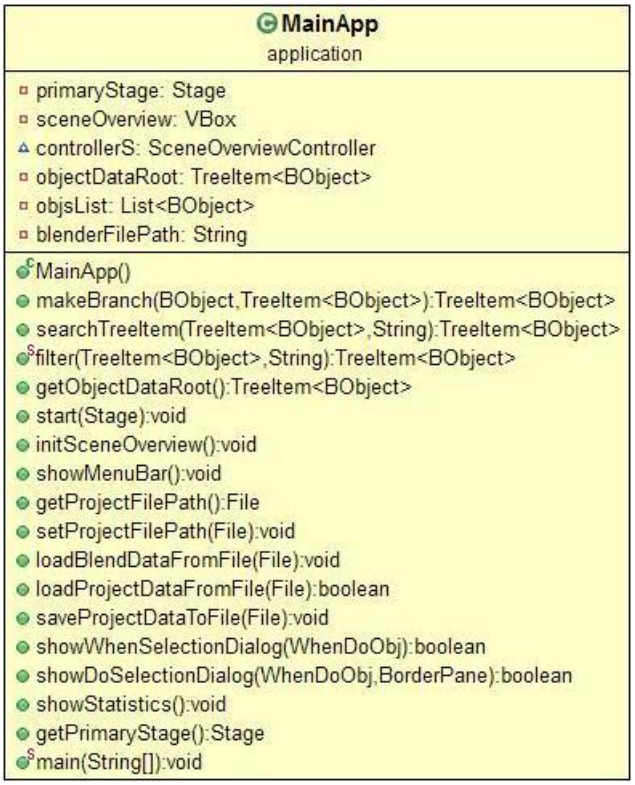


Figura 71. Dettaglio della classe MainApp.

- application.model: contiene le classi model. Quelle presenti nel progetto sono le tre strutture essenziali da gestire: “BScene” che corrisponde a una scena di Blender, “BObject” che rappresenta un oggetto di Blender interno a una scena e “WhenDoObj” che descrive un singolo blocco logico When, Do o global When (Fig.72). Inoltre è presente anche “ProjectWrapper”, che è una classe di supporto per la generazione del file XML finale.

  Una “BScene” possiede le informazioni di una scena creata dall’utente tra cui: il nome, la posizione nel pannello, il colore associato, se è la scena di partenza della simulazione, la lista degli oggetti posseduti, i link in ingresso e uscita da essa e i suoi When globali. Quindi i suoi metodi si occupano di settare e ottenere tali caratteristiche.

Nello stesso modo un elemento della classe “BObject” permette di memorizzare per un singolo oggetto qual è il suo nome, il suo eventuale oggetto padre, la posizione, la lista di animazioni presenti, l’immagine relativa a esso, la lista di blocchi When e Do posseduta e se tale oggetto sia o meno editabile, visibile o funga da puntatore. Un elemento “WhenDoObj” distingue tra un When globale a una scena, locale a un oggetto e un Do, indica l’interazione che possiede nel caso sia stato settato, un’immagine dell’icona visualizzata in esso, a cosa puntano gli eventuali link in uscita da esso e altre informazioni utili per una corretta gestione.

Figura 72. Dettaglio delle classi BScene, BObject e WhenDoObj.

Infine la classe “PojectWrapper” (Fig.73) possiede il path del file Blender aperto e la lista delle scene esistenti, in questo modo può predisporre la struttura di base del file XML da generare al salvataggio del progetto e quando l’utente vorrà salvare è sufficiente avviare un’operazione detta di marshalling su tale classe.

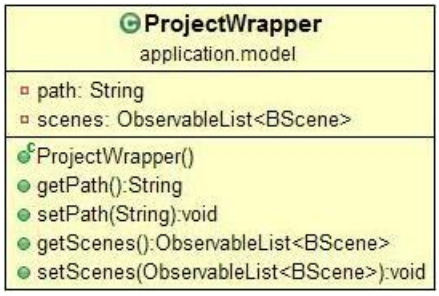


Figura 73. Dettaglio della classe ProjectWrapper.

- application.util: contiene le classi util. Il package util include “BlendLoader”, che viene utilizzato dal metodo *loadBlendDataFromFile()* precedentemente visto in “MainApp”, per ottenere gli oggetti di un file Blender iniziale. Per ricavare tali informazioni dal file, si è fatto ricorso alla libreria esterna “Liquidizer”, che permette di leggere i file “.blend” di Blender e li post-processa in una rappresentazione interna facilmente comprensibile, utilizzabile per esportarli in altri formati. Il metodo *LoadObjects()* ritorna una lista di classi oggetto di tale libreria, che si adatta poi alla personale classe “BObject”.
  Inoltre nel package util sono presenti gli adattatori “ColorAdapter”, “ImageAdapter”, “Point2DAdapter” (con “AdaptedPoint”) e “StringPropertyAdapter”, che si occupano tutti di estendere la classe “XmlAdapter” per i relativi casi, in modo da permettere una corretta conversione di tali classi in un formato che sia traducibile nell’XML da generare e anche viceversa, se si vuole riaprire un progetto precedentemente salvato (Fig.74).

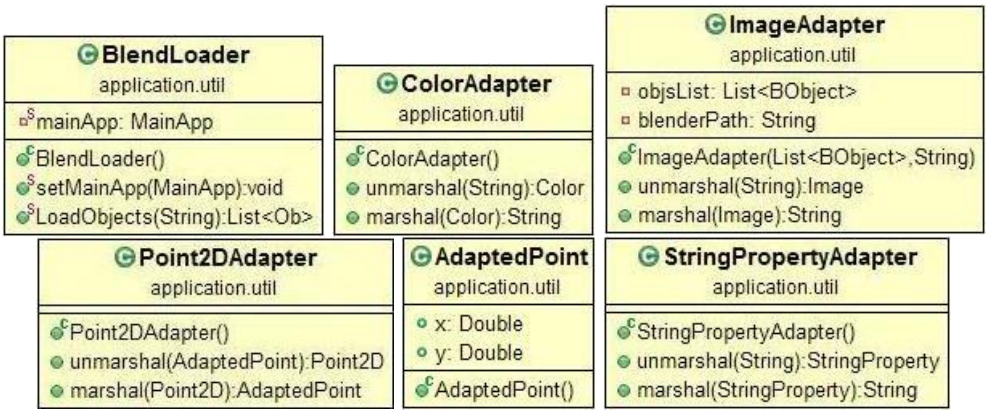


Figura 74. Dettaglio della classe BlendLoader e dei vari adattatori.

- application.view: contenente le view che definiscono l'aspetto visivo, sotto forma di file FXML. Inoltre il package view comprende tutti i controllori che sono relativi ad una singola view, siccome un requisito di JavaFX è che un controllore inerente a una view si trovi nello stesso package del relativo file FXML, affinché sia riconosciuta l'associazione. Questi controllori catturano gli eventi determinati dall'utente attraverso mouse e tastiera, traducendo queste azioni in cambiamenti di stato nella view.

  Tra i controllori, il più importante è "SceneOverviewController" (Fig.75), che è il file principale del progetto, in quanto gestisce il file FXML della finestra dell'intero programma. Trai suoi metodi si evidenziano *generateInitialScenes()* per creare le scene iniziali Schema e Scene_1 in un nuovo progetto, *handleOpenNewScene()* per aggiungere una nuova scena chiusa nel pannello generale, *showSceneEditDialog()* (con *showScene()*) per aprire una scena, *addObjToScene()* per aggiungere graficamente un oggetto nel pannello della scena aperta, *handleOk()* per salvare e chiudere una scena (chiamando prima *isInputValid()* per validare i dati della scena, poi *saveScene()* e infine *hideScene()* per chiuderla), *handleCancel()* per chiudere la scena aperta senza salvare, *handleOpenCopiedScene()* per copiare la scena aperta, *handleOpenNewNextScene()* per aggiungere una nuova scena successiva a quella aperta, *handleRemoveScene()* per rimuovere la scena aperta.

  La classe "MenuLayoutController" (Fig.77) si occupa della toolbar superiore dei menu, con un metodo per ciascun comando presente, come il salvataggio, l'apertura di un file Blender o di un progetto, la chiusura del programma, la finestra delle statistiche e quella di informazioni sul programma.

  "ButtonWD" (Fig.77) si occupa di un pulsante When o Do e dei link agganciati ad esso o che sono in corso di drag, "NodeLink" (Fig.75) gestisce la creazione, aggiornamento e rimozione dei link che connettono graficamente i When ai Do o alle scene, "WhenSelectionController" e "DoSelectionController" (Fig.76) si occupano della finestra di selezione di un nuovo When o Do e infine si ha "StatisticsController" (Fig.77) per la schermata delle statistiche del progetto. Al contrario dei pulsanti When e Do, le scene e gli oggetti non possiedono un relativo controller, ma i rispettivi file FXML "BScene.fxml" e "BObjInScene.fxml" sono completamente gestiti dalla classe "SceneOverviewController" generale, che quando richiesto ne aggiunge di nuovi e si occupa dei loro vari campi. Infine in questo package si segnala la presenza di un file CSS per definire alcuni dettagli grafici a certi elementi.

Figura 75. Dettaglio delle classi SceneOverviewController e NodeLink.

Figura 76. Dettaglio delle classi WhenSelectionController e DoSelectionController.

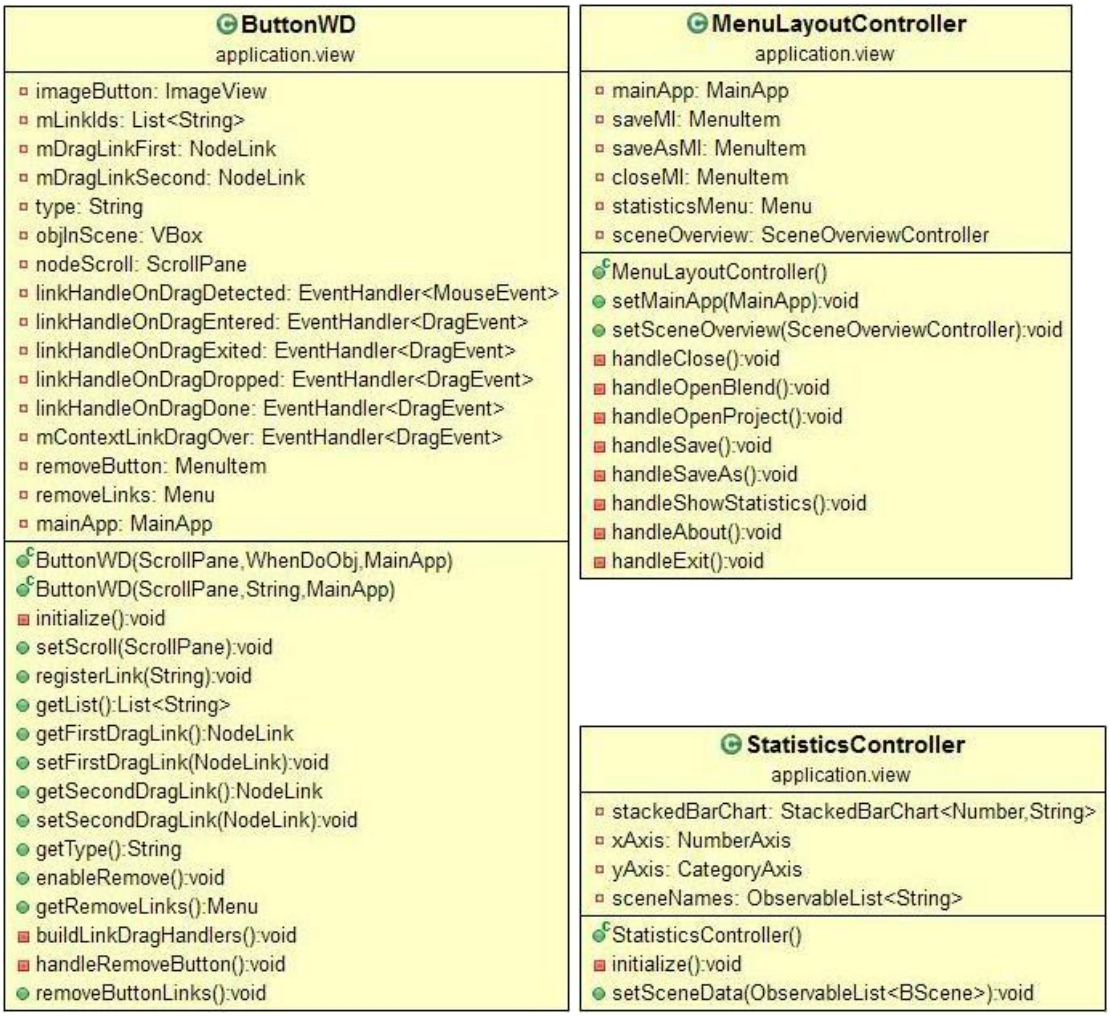


Figura 77. Dettaglio delle classi ButtonWD, MenuLayoutController e StatisticsController.

L'applicazione realizzata presenta l'architettura complessiva illustrata dal diagramma UML in Figura 78.

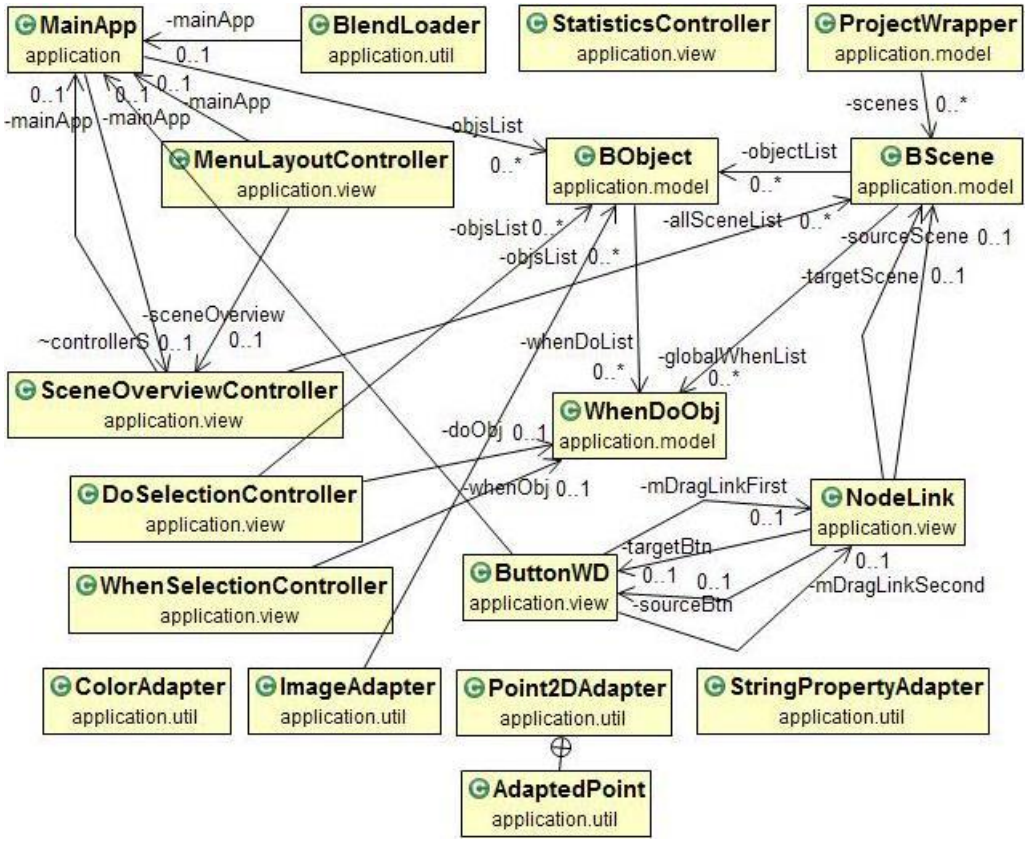


Figura 78. Diagramma UML delle classi del Visual Scene Editor.

4.1.5 Analisi migliorie

Per ottenere quei miglioramenti rispetto alla precedente versione del programma che si erano inizialmente prefissati come obiettivo della tesi, si sono seguite le buone norme di progettazione di un'applicazione, illustrate nel capitolo di analisi della progettazione dell'interfaccia, facendo altresì ricorso a numerosi accorgimenti, che semplificano la visione generale all'utente e velocizzano le operazioni.

Per esempio, il Leap Embedder non rendeva chiaro su quale scena si stesse lavorando, visto che richiedeva di definirle all'inizio spuntando gli oggetti interni, ma dopo si lavorava sui singoli oggetti senza più una visione chiara di come si comportassero nell'insieme tutti gli oggetti di una particolare scena.

Anche le interazioni tra scene non erano evidenti, mentre nel Visual Scene Editor si può subito capire a colpo d'occhio come esse si connettano tra loro, grazie al pannello principale che riunisce tutte le scene chiuse e mostra i loro collegamenti. Questa era una delle caratteristiche principali che si voleva ottenere, siccome per una persona è molto utile avere sempre una visione generale mentre lavora, sia di tutte le

scene sia di ciò che avviene dentro una singola di queste. Se prima si doveva aprire il pannello di un solo oggetto alla volta, ora si possono vedere contemporaneamente le interazioni di tutti gli oggetti di una scena, riuscendo a mantenere anche un aspetto pulito e compatto.
Le interazioni sono sempre composte da una coppia di condizione e azione (in questo caso chiamate When e Do), ma rispetto al riferimento non si disperdono più in un pannello richiedendo di distribuirle a mano in modo da essere visibili, ma restano ordinate in colonna semplificandone la gestione.
Un altro vantaggio dato dalla nuova impostazione è che, dato che prima si trascina l'oggetto nella scena e solo dopo si definisce la logica dell'oggetto per quella scena, si evita di definire logiche per oggetti che non sono presenti in nessuna scena, fatto che invece poteva verificarsi nella versione precedente.
Uno dei cambiamenti più evidenti apportati riguarda il concetto dei messaggi. Come si evidenzierà anche dai test eseguiti con utenti, l'applicazione Leap Embedder risultava poco intuitiva nella gestione dei messaggi, che rappresentavano la più frequente causa di errore. Questo perché lo strumento non astraeva il concetto di messaggio impiegato nel Logic Editor di Blender che si cercava di semplificare, ma obbligava a ricorrere ad una Action di tipo "Message" o "Multimessage" tutte le volte che un gesto scatenava una qualche conseguenza su un diverso oggetto. Il "Message" rifletteva l'omonimo attuatore del Logic Editor, necessitando di definirne un destinatario e un campo subject, mentre nell'oggetto ricevente si ricorreva a un blocco "WaitFor" che con il medesimo subject riceveva il messaggio che era stato inviato, per reagire con l'azione desiderata.
Nel realizzare la nuova applicazione si pretendeva di semplificare tale gestione e la soluzione adottata si è appurato essersi rivelata efficace e immediata per gli utenti: si sono infatti velocizzate le operazioni ricorrendo al posto dei messaggi ad una linea da collegare tra la condizione del When e l'azione del Do di un altro oggetto. In questo modo, si sono ridotte notevolmente le possibilità di generare errori, risparmiando tempo.
Secondariamente, si sono migliorati vari aspetti minori e dettagli, come ad esempio la definizione di un'animazione: se prima si doveva scrivere il nome corretto dell'animazione che andava ricordato, ora ogni oggetto mostra un menu a tendina con la lista di animazioni su di esso definite tra cui scegliere, impedendo di dichiarare un nome errato. Inoltre, non è più obbligatorio indicare i frame di inizio e fine animazione, ma lasciandoli non dichiarati verranno adattati automaticamente all'intervallo massimo. Anche l'indicazione della scena di partenza della simulazione è facilitato, in quanto la prima scena creata è per default impostata quale "start scene" (come probabilmente si desidera nella maggior parte dei casi).
Si è scelto di non includere più nell'applicazione la scelta della tipologia di visualizzazione tra teca olografica 3D e monitor 2D, in modo che un progetto realizzato non dipenda da essa, ma sia univoco, rimandando tale distinzione alla fase di importazione in Blender.
Infine, una gestione interna oculata delle funzioni messe a disposizione permette di evitare all'utente di generare errori imprevisti, come per esempio un conflitto tra due comandi o annullare un'azione

precedente in seguito a un'altra che la impedirebbe. Numerosi messaggi di errore o di semplice warning avvisano l'utente su possibili situazioni indesiderate che necessitano la sua attenzione.
Oltre a fare meglio ciò che era già possibile fare con la versione precedente, l'applicazione Visual Scene Editor ha potenziato anche le funzionalità disponibili rispetto alla precedente e rimosso alcune limitazioni e bug esistenti. I gesti manuali disponibili sono rimasti gli stessi, ma in aggiunta si ha un When di tipo OnLoad che permette di scatenare immediatamente delle azioni all'avvio di una scena; anche la precedente applicazione permetteva di ottenerlo, ma in modo meno logico sfruttando un messaggio da un oggetto di una scena a un altro oggetto della nuova scena a cui passare. Anche i Do sono stati tutti perfezionati, come le animazioni più intuitive accennate in precedenza o i suoni che ora accettano file audio di varie estensioni, e se ne sono aggiunti di nuovi come quelli del tab Object che permettono di cancellare, aggiungere o sostituire un oggetto.
Rispetto all'applicazione precedente, adesso è anche possibile assegnare azioni diverse a uno stesso oggetto in scene diverse. Per esempio un oggetto Cube può reagire a un KeyTap in un certo modo quando si trova nella scena 1 e in un modo differente quando è nella scena 2, questo grazie al fatto che ora, inserendolo in più scene, lo si gestisce come oggetti effettivamente indipendenti, anziché un univoco Cube per tutte le scene. Una tale gestione permette di creare simulazioni più complesse e in certi casi renderebbe sufficiente un numero molto inferiore di oggetti di partenza per ottenere lo stesso risultato.
Anche i When globali di una scena sono un concetto nuovo, e funzionano come se nella versione precedente si avesse avuto la possibilità di dichiarare un "GlobalObject" esclusivo a una sola scena.
L'aggiunta della scena Schema è servita invece a semplificare all'utente la gestione degli oggetti e delle interazioni sempre presenti. Si è cercato di velocizzare anche la gestione di scene identiche o quasi uguali introducendo un pulsante per duplicare la scena corrente creandone una copia avente gli stessi contenuti. Tutte queste funzionalità, combinate insieme, aumentano le potenzialità dell'applicazione, rivelandosi molto utili in certe situazioni.
Un altro aspetto che si è provveduto a migliorare è la colonna degli oggetti sulla destra, che ora mostra anche un'immagine di ciascuno, rendendo più semplice riconoscere a quale oggetto si sta facendo riferimento (per quanto si possano dare dei nomi intellegibili, solitamente non è facile riconoscere tutti gli oggetti solo dal loro nome). Adesso, ogni oggetto porterà con sé una rappresentazione grafica che aiuta l'utente a riconoscerlo senza nemmeno più dover per forza far riferimento al nome. Infine, gli oggetti nella colonna sono filtrabili e mostrati in un albero gerarchico, come in Blender, per riflettere le relazioni padre-figlio che li legano.

4.2 Importazione in Blender

Una volta completato il proprio progetto all'interno del programma, l'utente deve procedere a salvarlo in modo che sia successivamente importabile in Blender. Di seguito si descrive come avviene il salvataggio del progetto e come esso verrà alla fine rigenerato dentro Blender.

4.2.1 Salvataggio file XML del progetto

Per salvare il progetto realizzato, si deve selezionare dal menu l'opzione "Save" o "Save As…", che permette di ricavare un file XML, con il quale poter anche rigenerare il progetto in un secondo momento se si desidera rimetterci mano in seguito.
Per ottenere tale file, si ricorre alla libreria JAXB (Java Architecture for XML Binding) [70], che con poche linee di codice permette di generare un XML in output con tutte le informazioni necessarie. Attraverso un semplice comando di marshal, JAXB procede a serializzare un oggetto Java in XML, mentre con un unmarshal si effettua l'operazione inversa di conversione da un file XML alla corrispondente rappresentazione a oggetti Java.
Il mapping tra XML Schema e Java avviene grazie a opportune annotazioni: per esempio l'elemento radice del file è riconosciuto in quanto è quello che si ha opportunamente indicato come "@XmlRootElement", mentre un "@XmlElement" rappresenta un generico elemento e un "@XmlAttribute" è un attributo di un elemento.
Attraverso altre annotazioni si può specificare ad esempio in che ordine si desidera vengano elencati certi elementi, indicare di annidare una lista di elementi dentro a un altro oppure si può segnalare quali variabili di una classe non si vuole che vengano convertite in XML. Le annotazioni a disposizione sono numerose, grazie a esse si possono personalizzare i nomi salvati nel file XML e avere il pieno controllo sul file XML che verrà generato.
Le classi Adapter analizzate in precedenza servono in questa situazione, in quanto permettono a JAXB di gestire correttamente quelle classi particolari, che altrimenti non saprebbe convertire. Si ricorre a un'annotazione "@XmlJavaTypeAdapter" per segnalare che in un certo punto sia utilizzato un adattatore personale.
Come si era anticipato, per generare l'XML ci si appoggia ad un oggetto di classe "ProjectWrapper", al quale si passa tutto ciò che si vuole sia salvato, ovvero una lista delle scene coi loro oggetti e interazioni più il percorso del file Blender su cui si lavora. In questo modo è sufficiente dare un comando di marshal di tale oggetto per ottenere il file XML completo. Viceversa, se si desidera ricaricare un

progetto precedente, un comando di unmarshal leggerà il file XML restituendo un oggetto di classe ProjectWrapper.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <BlenderFileContent>
    <SourceFile>C:\Users\raniero\Desktop\Tesi Salaroglio\testUtenti.blend</SourceFile>
  - <Levels>
    + <Level name="Schema" isStart="false" color="0x000000ff">
    - <Level name="Start" isStart="true" color="0xf78fa7fc">
        <position y="49.0" x="384.0"/>
      - <GlobalWhenObjects>
        + <WhenDoObject whenOrDo="global When" mouseHandler="addWhenUpdatedOnAction" enabledRemove="false">
        </GlobalWhenObjects>
      - <Object name="Sphere" pointer="false" parent="" notVisible="false" editable="true">
          <Animations/>
          <position y="16.0" x="82.0"/>
          <image>C:\Users\raniero\Desktop\Tesi Salaroglio\testUtenti\Sphere.png</image>
        - <WhenDoObjects>
          - <WhenDoObject whenOrDo="When" type="KeyTap" tab="Gestures (Leap)" mouseHandler="addWhen3" enabledRemove="true">
            - <ScenesTargetedByLinks>
                <SceneTargetedByLink>Scene_2</SceneTargetedByLink>
              </ScenesTargetedByLinks>
            - <ObjsTargetedByLinks>
                <ObjTargetedByLink> </ObjTargetedByLink>
              </ObjsTargetedByLinks>
            - <IndexesButtonWDInObjTargetedByLinks>
                <IndexButtonWDInObjTargetedByLink> </IndexButtonWDInObjTargetedByLink>
              </IndexesButtonWDInObjTargetedByLinks>
            </WhenDoObject>
          + <WhenDoObject whenOrDo="Do" mouseHandler="addDoUpdatedOnAction" enabledRemove="false">
          + <WhenDoObject whenOrDo="When" mouseHandler="addWhenOnAction" enabledRemove="false">
          + <WhenDoObject whenOrDo="Do" mouseHandler="addDoOnAction" enabledRemove="false">
          </WhenDoObjects>
        </Object>
      + <Object name="Cube" pointer="false" parent="" notVisible="false" editable="true">
      </Level>
    + <Level name="Scene_2" isStart="false" color="0x377e55fc">
    + <Level name="Scene_3" isStart="false" color="0x385e41fc">
    </Levels>
</BlenderFileContent>
```

Figura 79. Esempio di file XML generato.

Il file XML ottenibile presenterà una struttura simile a quella mostrata in Figura 79. Dopo la riga iniziale indicante la versione di XML in uso e la codifica utilizzata, il tag “BlenderFileContent” funge da elemento radice. Contiene un tag “SourceFile” col percorso del file Blender su cui si è lavorato e un tag “Levels” che include tanti “Level” quante sono le scene create, ognuno con gli eventuali When globali, oggetti interni della scena e tutte le altre informazioni necessarie a poter ricreare un progetto. La rappresentazione gerarchica dei tag aiuta in questo senso a capire che una certa scena possiede al suo interno determinati oggetti, i quali a loro volta hanno una loro lista di When e Do.

4.2.2 Apertura file XML in Blender

Una volta che si è ottenuto un file XML del progetto generato nel programma Visual Scene Editor, non resta che importarlo in Blender per ottenere concretamente le scene volute.

Per fare questo, si ricorre al Visual Scene Importer: un add-on Blender di importazione che si è creato, il quale procede a leggere un file XML desiderato e genera le scene in esso contenute, creando nel Logic

Editor di Blender tutti i mattoni logici necessari a gestire correttamente le interazioni desiderate. In realtà si dispone di due differenti versioni del Visual Scene Importer tra cui scegliere: una versione pensata per le interazioni con il Leap Motion e la teca olografica e una rivolta all'utilizzo su monitor con comandi di mouse e tastiera. L'utente perciò, in base a come intende visualizzare le scene, sceglierà di generarle con uno dei due importer.

Una volta che questi add-on sono installati (come illustrato in Figura 80), viene aggiunto un nuovo comando all'interno dell'interfaccia di Blender per ciascuno di essi, attraverso il quale invocarli.

In sintesi le due principali differenze che contraddistinguono la versione per il Leap Motion da quella per la tastiera sono appunto il tipo di input su cui rilevano le interazioni (gesti della mano piuttosto che comandi da tastiera) e che nella versione per la teca olografica l'importer procede a generare tre camere differenti che inquadrino la scena corrente dai vari lati in modo da creare così l'effetto ologramma, mentre nella versione per monitor 2D si avrà una sola camera frontale.

Nel caso dell'importer che permette di interagire con il Leap Motion, l'interfacciamento con il dispositivo è stato ripreso dal precedente lavoro di tesi. Perciò, in questa sede non ci si sofferma a illustrarne il funzionamento. In sostanza, affinché il Blender Game Engine possa riconoscere i gesti effettuati con le mani, l'importer provvede a richiamare altri script Python adibiti all'interfacciamento col Leap e al riconoscimento dei gesti, attraverso la generazione di controllori Python nel Logic Editor. Quando viene rilevato un particolare gesto eseguito dall'utente (tra KeyTap, ScreenTap, Swipe o Circle), nel Logic Editor si genererà un attuatore di tipo Message che provvederà ad attivare l'opportuno sensore Message relativo a tale gesto. Non si è praticamente rivelato necessario intervenire su tali script preesistenti di riconoscimento dei gesti del Leap Motion, salvo irrilevanti accorgimenti.

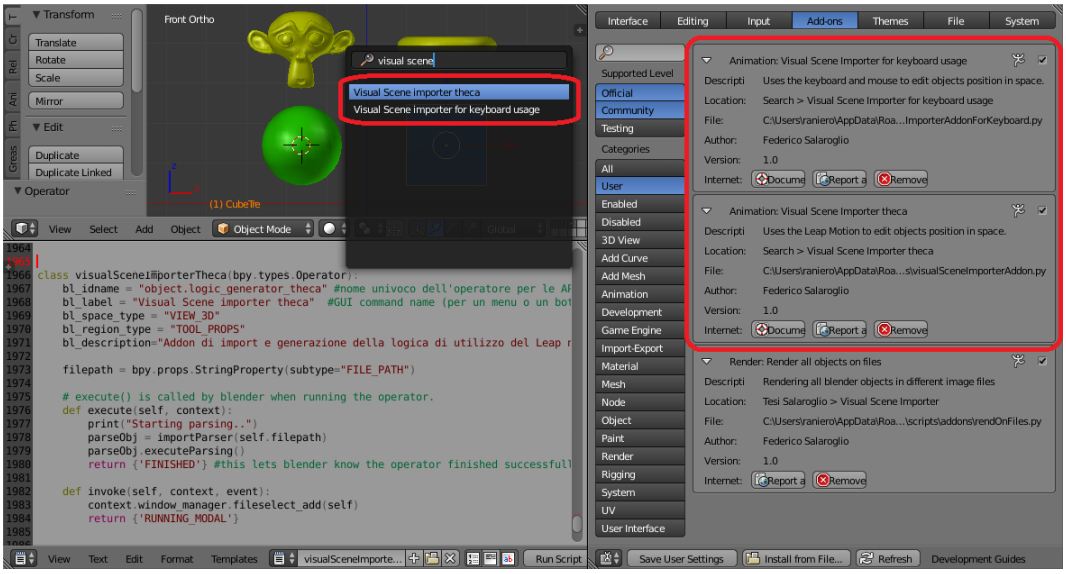


Figura 80. Installazione degli add-on dell'importer.

Indifferentemente da quale importer si utilizzi, questo come prima azione chiederà all'utente di selezionare il file XML che si intende importare, per poi poter procedere ad analizzarlo in modo da ricreare la struttura di scene indicata. Una volta che l'utente ha indicato un file XML, viene verificato che esso sia effettivamente un file XML correttamente formattato e che faccia riferimento al file Blender correntemente aperto. A questo punto si procede dapprima a pulire ogni oggetto presente da eventuali mattoni logici e proprietà di gioco esistenti in esso, in modo da evitare che non restino costrutti logici precedenti, non desiderati rispetto alla simulazione che si vuole ottenere. Inoltre per precauzione si applica ogni modificatore eventualmente presente negli oggetti, altrimenti nel copiarli in più scene si potrebbero rilevare problemi interni a Blender come il non funzionamento dello shading smooth sugli oggetti delle scene finali.

Una volta che si è preparato l'ambiente, si procede a creare tutti quei componenti che sono obbligatori in ogni scena per un corretto funzionamento. Si definisce la gestione del puntatore, creando un oggetto non visibile "rayEmitterObject" che seguirà sempre il puntatore; quest'oggetto attraverso un sensore Ray proietta un raggio invisibile in profondità e rileva se il suo raggio collide su un qualche oggetto per determinare su quale oggetto è posizionato l'elemento puntatore, comunicando a quest'ultimo qual è l'oggetto attivo attraverso un controllore di tipo Python. L'elemento puntatore avrà una property "activeObject" in cui memorizzare il nome dell'oggetto correntemente puntato, così saprà sempre quale degli oggetti della scena detiene il focus. Inoltre, si creano le camere (o la singola camera), impostandone i parametri di configurazione e rendendole anche spostabili via tastiera a simulazione in corso.

A questo punto si può finalmente procedere a creare tutte le scene richieste, popolandole con i relativi oggetti. La scena Schema non deve essere creata, non essendo una scena effettiva, ma i suoi oggetti verranno inseriti in tutte le altre scene. In ogni scena vengono anche impostati i parametri di illuminazione del "World", per sopperire al caso in cui l'utente non avesse provveduto a inserire delle fonti di illuminazione. Inoltre alle scene si inizializzano opportunamente i parametri di rendering e della configurazione del motore di simulazione fisica. Si segnala che tutti gli oggetti di una scena (ad eccezione di camere e "rayEmitterObject") verranno rinominati anteponendo al loro nome il nome della scena in cui si trovano, in modo da mantenere dei nomi univoci agli oggetti anche quando essi vengono distribuiti su più scene.

Per l'oggetto impostato come puntatore della scena vengono creati i mattoni logici di tracciamento: nel caso del Leap quest'oggetto seguirà la posizione rilevata della mano dell'utente, mentre nel caso della tastiera simulerà a schermo gli spostamenti del mouse. Il programma Visual Scene Editor obbliga l'utente a indicare un puntatore in ogni scena, ma per sicurezza nel caso non si rilevi comunque nessun puntatore in una scena del file XML l'importer provvede a generare un oggetto di default che funga da puntatore, assegnando a esso la logica di tracciamento.

In ogni scena si aggiungono anche le tre camere (o la camera) e il “rayEmitterObject” creati in precedenza, assieme a un ulteriore oggetto “globalEmptyObject” non visibile di tipo Empty. A quest’ultimo si associano le condizioni specificate nei When globali della scena e inoltre gestisce anche dei messaggi per tutte le interazioni di ogni oggetto della scena.
Per spiegare la procedura completa con cui viene convertita l’interazione su un oggetto in mattoni logici del Logic Editor si può illustrare il funzionamento con un esempio. Se l’utente avesse dichiarato ad alto livello che si deve poter rilevare un KeyTap su un oggetto Cube, allora prima di tutto si riceverà un Message di KeyTap rilevato al “globalEmptyObject”, questo possiede una property “puntatore” in cui è memorizzato qual è l’oggetto che funge da puntatore e grazie a questa proprietà può inoltrare il Message di KeyTap al puntatore. Il puntatore, ricevuto tale Message, consulta la sua property “activeObject” e se essa indica che in quell’istante si stava puntando l’oggetto Cube desiderato allora si ha effettivamente rilevato un KeyTap su di esso e il Message di KeyTap viene infine mandato all’oggetto. Quest’ultimo riceve il Message con un sensore e un attuatore provvederà a scatenare su di lui l’azione indicata nel Do conseguente al When (o più di un’azione in caso di Do multipli). Si ricorda che anche i cambi di scena vengono realizzati con un attuatore opportuno. Invece quando si ha che un When su un primo oggetto scatena un Do di un altro oggetto, l’attuatore dell’oggetto iniziale si limita a mandare un Message al secondo oggetto che lo riceve via sensore e provvede a realizzare il Do con un suo attuatore.
Una volta che si sono realizzate nel Logic Editor le interazioni dei global When della scena e le interazioni di ogni singolo oggetto presente in essa, si ripete il tutto con i global When e gli oggetti della scena Schema. Così se in Schema fosse presente un’interazione che entrerebbe in conflitto con un’altra specifica della scena la si omette, in quanto la regola di fondo che si ha scelto di seguire è quella di far prevalere una dichiarazione più specifica a una più generica.
Per ultimo si definisce la visibilità degli oggetti e si ripristinano eventuali imparentamenti tra oggetti ove presenti.
Al termine della definizione di tutte le scene viene eliminata la scena di default inizialmente presente in Blender e si imposta come scena di partenza della simulazione quella che risulta indicata come tale nel file.
Viene così completata la generazione della logica di interazione e all’utente non rimane che avviare la simulazione o, se lo desidera, apportare personalmente ulteriori aggiustamenti, ad esempio all’illuminazione o alla disposizione degli oggetti.

Capitolo 5

Risultati

Illustrato il lavoro di tesi realizzato, si procede a esporre i risultati ottenuti. Si inizia con una prima valutazione del Visual Scene Editor attraverso la realizzazione di una simulazione centrata su una rappresentazione virtuale del busto della regina Nefertiti pensata, ad esempio, per esposizioni museali. Tale simulazione è stata considerata anche in precedenti lavori di tesi, e viene ripresa in quanto abbastanza strutturata da coinvolgere un ampio spettro di funzionalità da gestire, e quindi utile a valutare, almeno in prima istanza, i risultati conseguiti. A seguire, vengono invece analizzati i risultati di esperimenti di usabilità condotti con volontari su scene di test al fine di valutare in maniera quantitativa se siano stati raggiunti quegli obiettivi inizialmente prefissati per il lavoro.

5.1 Il caso di studio Nefertiti

Si è scelto di utilizzare come progetto Blender di riferimento lo stesso esaminato nel precedente lavoro di tesi nel quale era stato realizzato il Leap Embedder, realizzato nel lavoro di tesi per il corso di laurea in Design e Comunicazione Visiva. Trattasi di una simulazione 3D per la teca olografica relativa al busto della regina egizia (Fig.81), alto circa 50cm e realizzato in pietra calcarea durante il regno del faraone Akhenaton tra il 1353 e il 1336 a.C., che dal 2009 è l'attrazione principale del Neues Museum di Berlino [71].

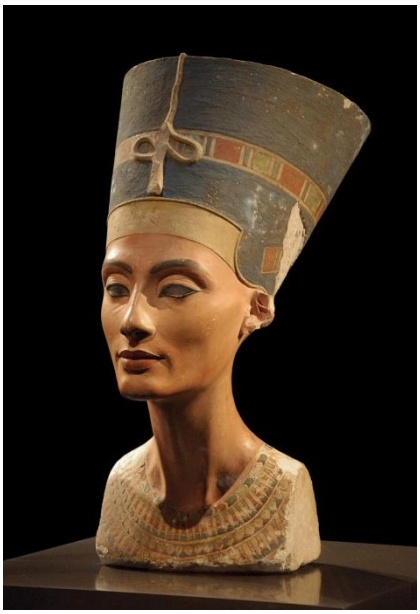


Figura 81. Busto di Nefertiti, Berlino. (fonte: wikipedia.org)

L'applicazione realizzata consiste in una riproduzione del busto attraverso un insieme di scene che ne illustrano i dettagli. La simulazione era stata realizzata pensandola per un ambito museale, dove un visitatore può interagire con essa, in questo caso sia attraverso uno schermo classico che per mezzo dei gesti delle mani se all'interno di una teca olografica. Così, l'utente può osservare il busto da varie angolazioni, leggendone la storia e varie descrizioni, in un modo più coinvolgente, secondo una concezione di exhibit dinamico. Inoltre, si può parlare anche di exhibit sociale, in quanto più visitatori possono alternarsi nelle interazioni e osservare contemporaneamente la simulazione da varie angolazioni, riunendosi in gruppo in un momento ludico-istruttivo che può stimolare l'interesse verso l'opera in questione.

5.1.1 Progetto dell'interazione

Il punto di partenza è ovviamente quello di possedere un file Blender con il busto e tutti gli altri elementi modellati, le animazioni che si desidera gestire già create e tutto quello che servirà visualizzare nelle scene finali (Fig.82). In questo caso, tra i vari elementi non sono presenti le luci, che risulteranno assenti nelle varie scene, relegando l'illuminazione finale ai parametri di illuminazione del "World". Le tre camere per la teca olografica sono invece presenti per capire quale sia il volume di vista (o frustum) centrale anche se non fondamentali, dato che l'applicazione Visual Scene Editor non le gestisce e l'importer avrebbe esso stesso provveduto a crearle se non le avesse rilevate.

Nel file Blender tutti gli oggetti devono risiedere in un'unica scena, ma possono essere distribuiti a piacimento nei 20 layer disponibili. Si segnala che è sempre consigliabile assegnare nomi facilmente riconoscibili agli oggetti ed alle animazioni, in modo da semplificarne il riconoscimento nell'applicazione Visual Scene Editor. Sempre riguardo al semplificare il riconoscimento degli oggetti, l'esecuzione dello script Python "rendOnFiles.py" sul file Blender permetterà comunque di disporre nell'applicazione di un'immagine per ogni elemento.

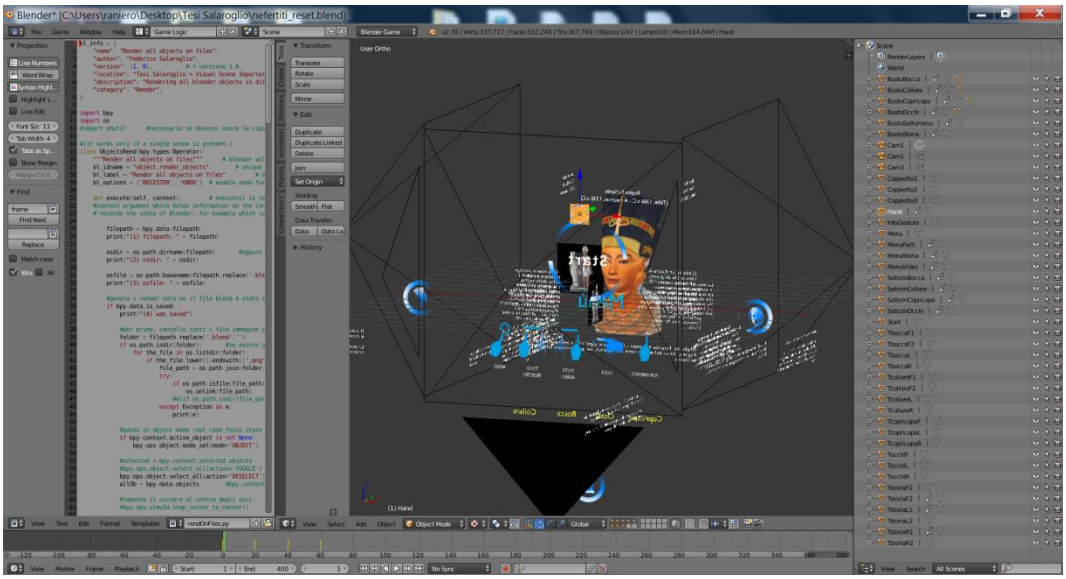


Figura 82. File Blender Nefertiti di partenza preimpostato.

Una volta assicuratisi di possedere il file Blender correttamente preimpostato, si può avviare l'applicazione Visual Scene Editor e caricare il file selezionando da menu "Open .blend". Il file verrà elaborato e al termine ci si ritroverà con una prima scena vuota aperta e con la colonna di destra riempita di tutti gli oggetti presenti nel file. Come si può notare in Figura 83, alcuni oggetti possiedono un altro oggetto figlio al loro interno, come "BustoBocca" che contiene "BustoBocca2", così come risultava in Blender.

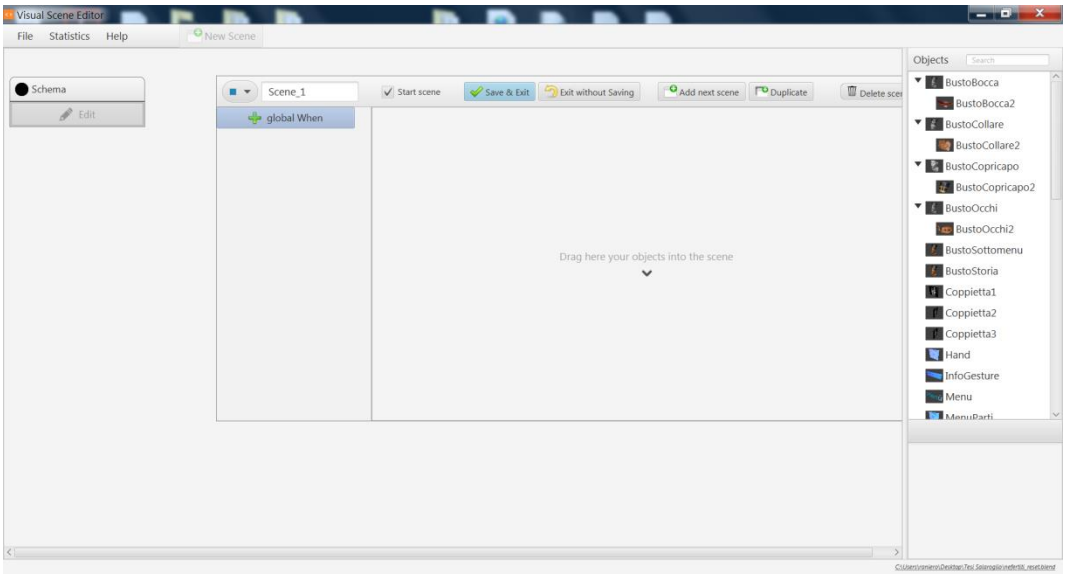


Figura 83. Interfaccia dell'applicazione inizializzata.

A questo punto si può procedere nel realizzare la simulazione desiderata. La simulazione si comporrà di una vista iniziale da cui partire, in cui verranno illustrati i gesti manuali con cui poter interagire e si resterà in attesa di un primo comando da parte dell'utente. Da qui, se si rileva che una persona ha eseguito un gesto di tipo KeyTap, si passerà a un menu di navigazione, da cui poter accedere alle varie sezioni che illustreranno una serie di approfondimenti e contributi relativi al busto: si potrà visualizzare una vista del busto che include una descrizione testuale della sua storia o passare a un altro menu in cui selezionare altre quattro viste relative ad altrettanti dettagli estetici del busto. In qualsiasi momento, si può fare ritorno al menu principale con un semplice gesto Circle, così si può raggiungere ogni sezione comodamente. Invece le interazioni che un utente farebbe solitamente con un clic del mouse sono associate alla gesture KeyTap, che è quella intuitivamente più vicina. Inoltre, ai diversi oggetti sono state assegnate funzioni logiche, temporizzazioni, animazioni e altre caratteristiche in grado di rendere il risultato più dinamico. Ogni vista può essere liberamente navigata dall'utente che è situato di fronte alla teca, mentre chi osserva dai lati potrà vedere ugualmente tutto ciò che accade dalla sua angolazione.
Le scene che si sono realizzate nell'applicazione sono le seguenti:

- *Start*: la scena iniziale visualizzata in attesa che un utente avvii la simulazione; pertanto, nell'applicazione la si deve spuntare quale scena di partenza;
- *Menu*: scena centrale contenente un menu attraverso cui poter accedere alle varie macro-sezioni della simulazione;
- *Storia*: sezione con il busto della regina Nefertiti e descrizione testuale contente informazioni storiche;

- *MenuPezzi*: sotto-menu da dove poter accedere a propria scelta alle quattro scene di descrizione dei dettagli del busto;
- *pagBocca*: scena che mette in risalto la bocca del busto di Nefertiti, con descrizione testuale;
- *pagCollare*: scena che evidenzia il collare di Nefertiti, con descrizione testuale;
- *pagCopric*: scena che evidenzia il copricapo della regina, con descrizione testuale;
- *pagOcchi*: scena che mette in risalto gli occhi di Nefertiti (dove l'occhio sinistro è assente ed è stato dimostrato che non venne mai stato realizzato), con descrizione testuale;

Dichiarate le scene e inseriti tutti gli oggetti necessari in esse, resta da definire la logica di interazione per ogni oggetto e gli opportuni When globali nelle varie scene. La scena "Schema" che vale per tutte le scene avrà per esempio un When globale che indica di ricaricare la scena "Menu" ogni volta che si rileva un Circle, indipendentemente dalla scena corrente, mentre in questa scena speciale si può inserire l'elemento "Hand" che funge da puntatore in tutte le scene. Invece le scene "Menu", "MenuPezzi" e le quattro rappresentanti i dettagli del busto avranno tutte le icone relative alle scene a cui poter passare che all'avvio entrano nell'inquadratura dai vari lati, grazie a un'animazione assegnata a ciascuna di esse che si avvia in concomitanza con il caricamento della rispettiva scena. In modo analogo, le scene "Storia", "MenuPezzi" e le quattro scene dei dettagli del busto avranno il busto di Nefertiti che all'avvio della scena inizia a ruotare lentamente su sé stesso, per poterlo osservare interamente. Per realizzare tutte queste animazioni all'avvio di tali scene, si può applicare un When globale alle scene di tipo OnLoad che quindi si attiverà subito al caricamento della scena, come mostrato in Figura 84.

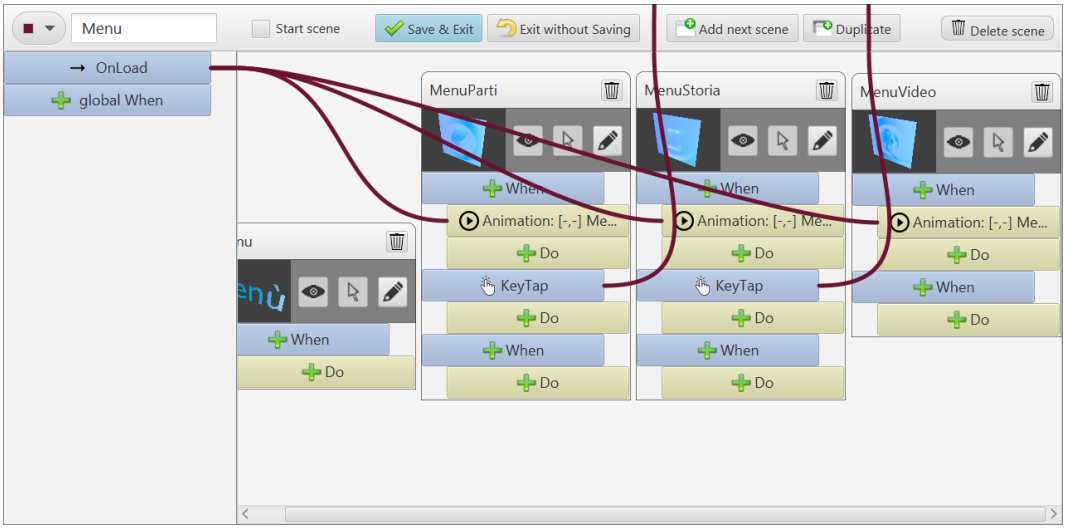


Figura 84. Scena "Menu" con un When globale di tipo OnLoad che scatena animazioni all'avvio nella scena.

In modo analogo si specificano tutte le interazioni degli oggetti attraverso i loro comandi When e Do, per esempio indicando che a un KeyTap su di un particolare oggetto si scateni il passaggio a una nuova

scena, collegamento che nell'applicazione viene rappresentato da una linea che parte dal When di tipo KeyTap dell'oggetto e termina alla scena esterna.

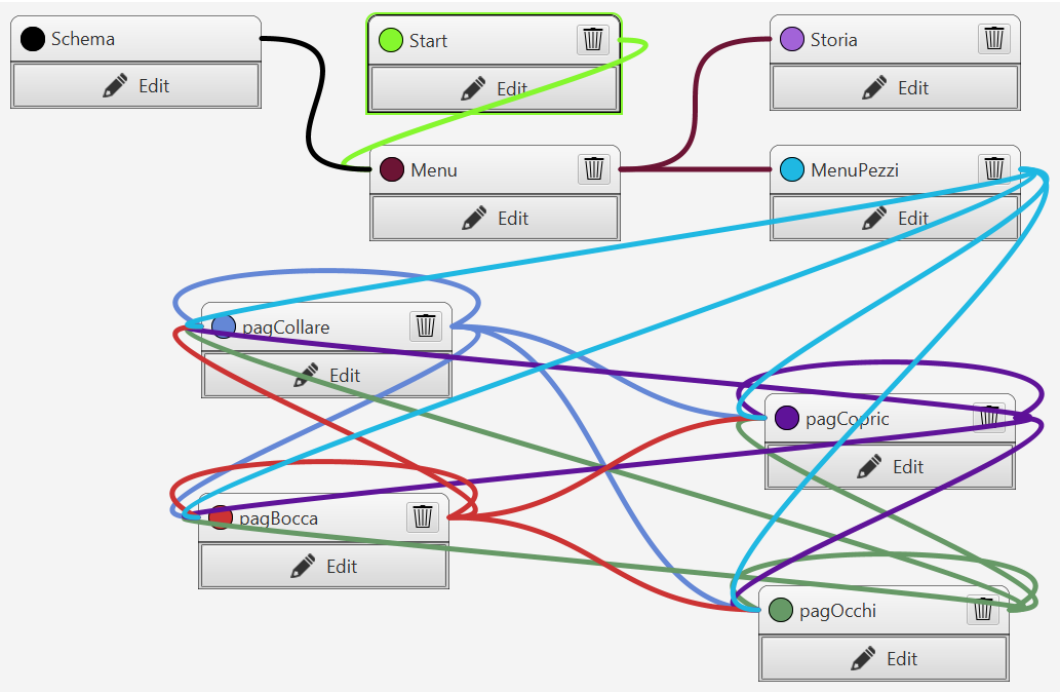


Figura 85. Le varie scene chiuse e i collegamenti tra esse.

Terminata la definizione della logica all'interno dell'applicazione (Fig.85), si può generare il file XML di esportazione che includerà le otto scene più quella "Schema" (Fig.86), ognuna con i suoi oggetti interni e interazioni (Fig.87).

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <BlenderFileContent>
    <SourceFile>C:\Users\raniero\Desktop\Tesi Salaroglio\nefertiti_reset.blend</SourceFile>
  - <Levels>
      + <Level name="Schema" isStart="false" color="0x000000ff">
      + <Level name="Start" isStart="true" color="0x84f72bfc">
      + <Level name="Menu" isStart="false" color="0x6b1232fc">
      + <Level name="Storia" isStart="false" color="0xa162d8fc">
      + <Level name="MenuPezzi" isStart="false" color="0x1db8e2fc">
      + <Level name="pagCollare" isStart="false" color="0x6386d6fc">
      + <Level name="pagBocca" isStart="false" color="0xcc3333ff">
      + <Level name="pagCopric" isStart="false" color="0x5e1198fc">
      + <Level name="pagOcchi" isStart="false" color="0x669966ff">
    </Levels>
</BlenderFileContent>
```

Figura 86. File XML del progetto con i tag chiusi delle varie scene.

```
- <Object name="Start" editable="true" pointer="false" parent="" notVisible="false">
    <Animations/>
    <position y="45.0" x="78.0"/>
    <image>C:\Users\raniero\Desktop\Tesi Salaroglio\nefertiti_reset\Start.png</image>
  - <WhenDoObjects>
    - <WhenDoObject whenOrDo="When" type="KeyTap" tab="Gestures (Leap)" mouseHandler="addWhen3" enabledRemove="true">
      - <ScenesTargetedByLinks>
          <SceneTargetedByLink>Menu</SceneTargetedByLink>
        </ScenesTargetedByLinks>
      + <ObjsTargetedByLinks>
      + <IndexesButtonWDInObjTargetedByLinks>
      </WhenDoObject>
    + <WhenDoObject whenOrDo="Do" mouseHandler="addDoUpdatedOnAction" enabledRemove="false">
    + <WhenDoObject whenOrDo="When" mouseHandler="addWhenOnAction" enabledRemove="false">
    + <WhenDoObject whenOrDo="Do" mouseHandler="addDoOnAction" enabledRemove="false">
    </WhenDoObjects>
  </Object>
```

Figura 87. Dettaglio dell'oggetto Start nella scena omonima, in cui un KeyTap sull'oggetto genera un passaggio a scena "Menu".

5.1.2 Importazione in Blender

Terminato l'utilizzo dell'applicazione Visual Scene Editor, si importa in Blender il file generato. Si sceglie se usare l'add-on di importazione per la teca 3D o per monitor 2D e, una volta avviato, questo replicherà la logica di interazione all'interno del motore di gioco.

Il risultato che si ottiene (Fig.88) è che se prima si aveva una sola scena di default contenente tutti gli oggetti, ora si disporrà di tutte le scene create e gli oggetti saranno correttamente distribuiti in esse secondo quanto definito dall'utente (Fig.89). Se si seleziona un oggetto, si potrà notare come questo avrà nel Logic Editor tutti i mattoni logici adibiti alla gestione delle relative interazioni (nella 3D View le scritte e le immagini risulteranno capovolte, siccome nel proiettarle sulla teca olografica verranno ribaltate e visualizzate in maniera adeguata). Inoltre l'importer per la teca avrà provveduto a creare il sistema delle tre camere, come in Figura 90, per la visione olografica (Fig.92).

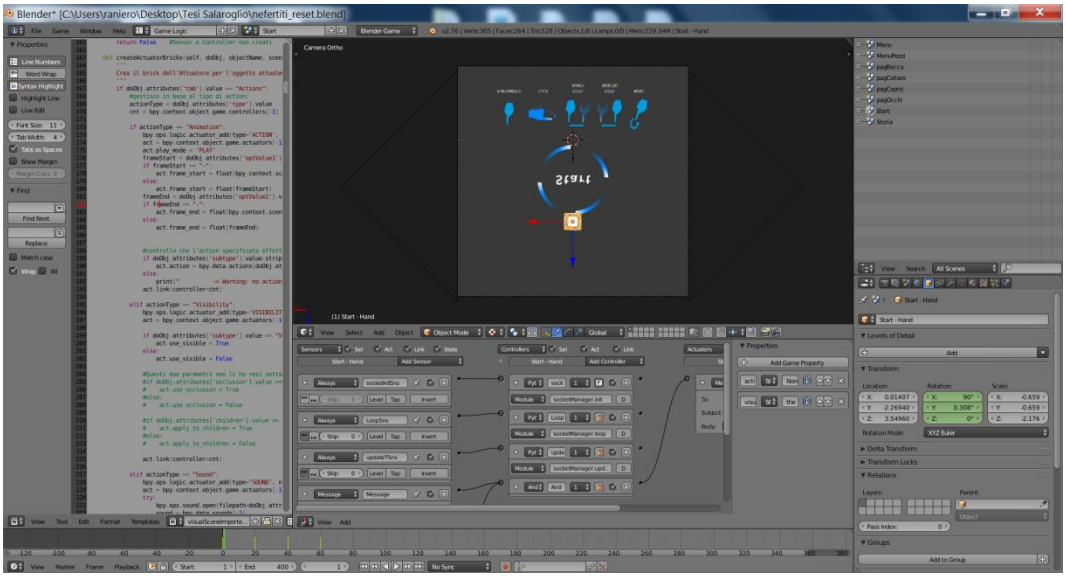


Figura 88. Risultato dell'esecuzione dell'importer.

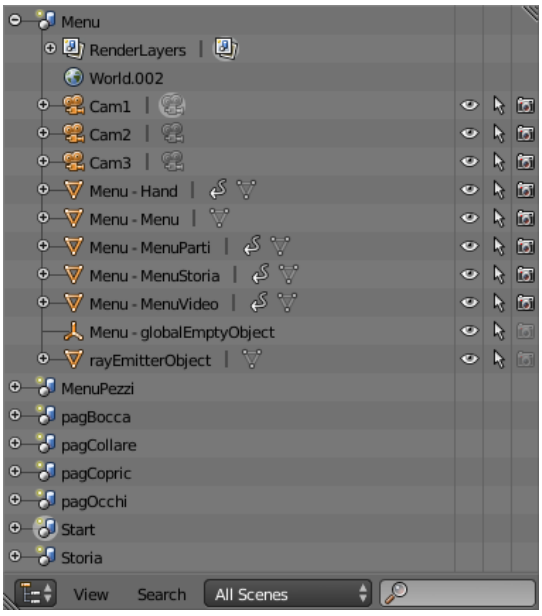


Figura 89. Outliner di Blender con la lista delle scene ricreate.

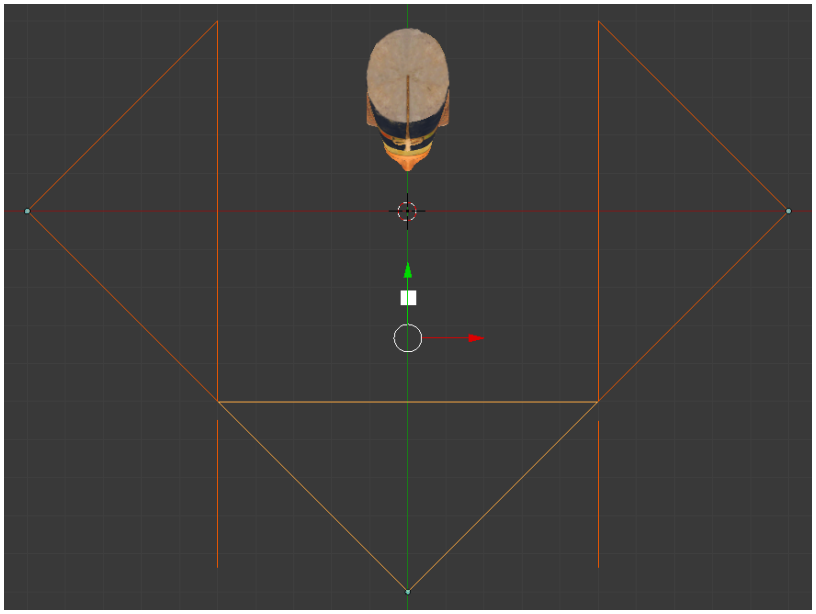


Figura 90. Dettaglio della 3D View sul sistema delle tre camere generato per la teca olografica.

Ora che si è creata la simulazione, la si può avviare in Blender (Fig.91). Il Blender Game Engine permette di scegliere tra “Embedded Player” e “Standalone Player” per avviare il progetto, ma in alternativa è anche possibile esportarne un file eseguibile.

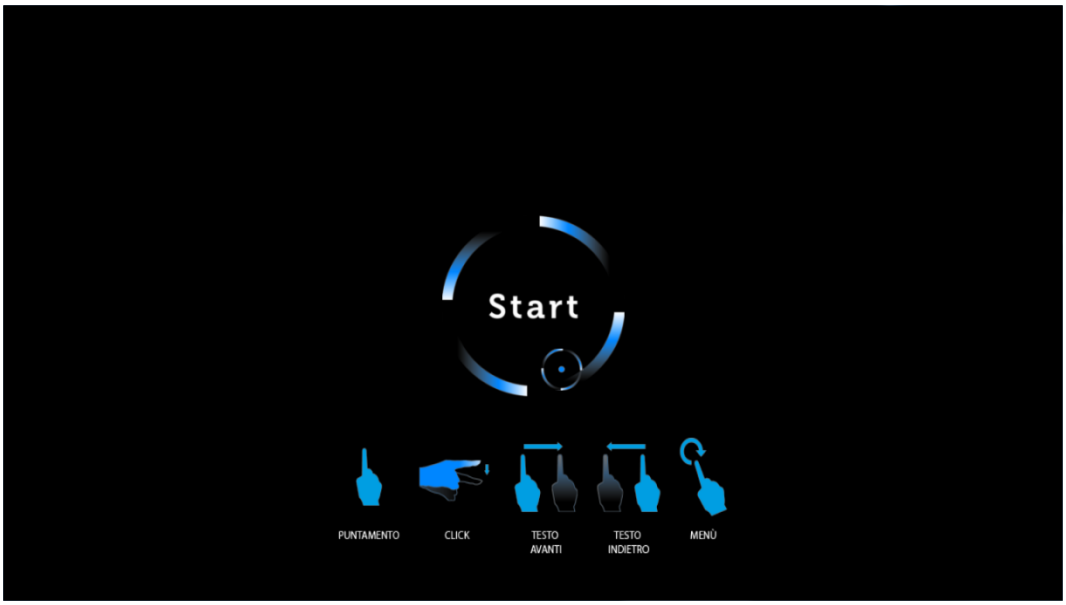


Figura 91. Scena "Start" di avvio della simulazione.

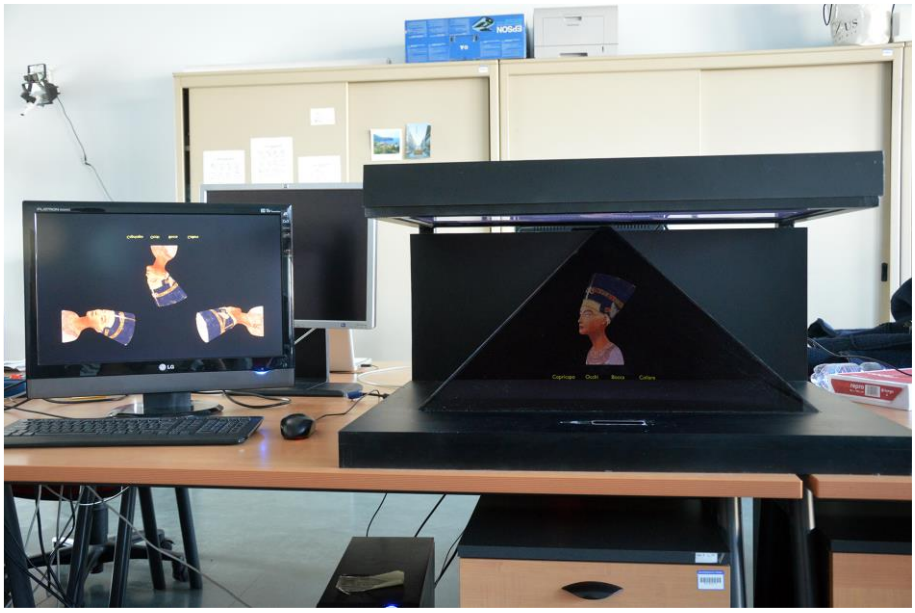


Figura 92. Teca olografica con la scena "MenuPezzi". (fonte: tesi di Laurea Magistrale in Ingegneria Informatica di Luigi Maggio)

5.2 Test con utenti

Per poter valutare la bontà dei risultati ottenuti dal lavoro di tesi, si è confrontata la nuova applicazione con quella precedente, in modo da verificare se si sono effettivamente ottenuti dei miglioramenti e di che portata.

Lo scopo del progetto era quello di ottenere un riscontro del fatto che il Visual Scene Editor porti dei benefici rispetto al Leap Embedder, ovviamente accertandosi in partenza che quest'ultimo risultasse effettivamente vantaggioso rispetto all'utilizzo diretto di Blender, che deve essere una caratteristica fondamentale da cui poter avviare ogni sorta di analisi.
Ci si aspetta, in particolare, di ottenere un'intuitività maggiore, sia rispetto al Leap Embedder sia rispetto al Logic Editor di Blender, unita ad una maggiore velocità di esecuzione del lavoro e ad una visione più chiara di ciò che si sta gestendo e realizzando.
Il modo migliore per valutare il raggiungimento di tali obiettivi attraverso un giudizio oggettivo è quello di realizzare dei test di usabilità con utenti, in quanto una persona esterna e senza alcuna competenza in questi campi è il candidato ideale a cui la nuova applicazione si rivolge.
Si è proceduto pertanto a far utilizzare la nuova applicazione a un certo numero di persone, spaziando tra chi ha una qualche conoscenza di Blender a chi non ha particolari competenze in ambito informatico.
Ad ognuna di queste persone si ha fatto svolgere uno stesso test che non risultasse troppo complicato, ma comunque abbastanza strutturato da richiedere dei ragionamenti e l'utilizzo delle principali funzionalità e costrutti di interazione.
Si ha chiesto ad una serie di volontari di svolgere il test sia con il nuovo Visual Scene Editor che con il Leap Embedder, in modo da poter eseguire al termine un confronto dei risultati. Ad ogni volontario si è prima illustrato lo scopo del lavoro di tesi ed esposto il funzionamento dei due programmi, per un'introduzione complessiva di circa 20 minuti, rispondendo a eventuali domande relative a dubbi sulla modalità di utilizzo degli strumenti.
Si è scelto di non far ricreare la simulazione direttamente nel Blender Game Engine, neanche a soggetti eventualmente esperti con tale programma, in quanto il test proposto, anche se non particolarmente complesso, una volta importato nel BGE genera già un numero significativamente elevato di mattoni logici (tra sensori, controllori, attuatori e proprietà) che richiederebbe molto tempo per essere realizzato.
Agli utenti si ha fornito il file Blender visibile in Figura 93, che possiede dei semplici oggetti modellati.
Il test consiste di tre scene da ricreare:

- "Start" in cui sono presenti i quattro oggetti *Lamp* e *pointer*, *Sphere* e *Cube*;
- "Scene_2" in cui sono presenti i quattro oggetti *Lamp* e *pointer*, *SuzanneDue*, *TorusDue* e *SphereDue*;
- "Scene_3" in cui sono presenti i quattro oggetti *Lamp* e *pointer*, *SuzanneTre*, *TorusTre* e *CubeTre*.

L'interazione che è stato richiesto di modellare è invece la seguente:

- designare l'oggetto *pointer* quale puntatore in tutte le scene;

- designare la scena “Start” quale scena di partenza della simulazione;
- fare in modo che una gesture Circle produca sempre una transizione alla scena “Start”, indipendentemente dalla scena correntemente impostata;
- in “Start”:
    - un KeyTap sull’oggetto *Sphere* produca una transizione alla “Scene_2”;
    - un KeyTap sull’oggetto *Cube* produca una transizione alla “Scene_3”;
- in “Scene_2”:
    - una Swipe con direzione destra sull’oggetto *SuzanneDue* scateni sia su di esso l’inizio di un’animazione denominata “Suzanne2Action” sia in *TorusDue* cambii la visibilità a non visibile;
    - sia un KeyTap sull’oggetto *SphereDue* sia un KeyTap su *TorusDue* scatenino l’inizio di un’animazione dell’oggetto *SuzanneDue* denominata “Suzanne2Action”;
- in “Scene_3”:
    - un KeyTap su *CubeTre* scateni l’inizio di un’animazione dell’oggetto *SuzanneTre*, denominata “Suzanne3Action”;
    - un KeyTap sull’oggetto *SuzanneTre* scateni l’inizio di un’animazione dell’oggetto *CubeTre* denominata “Cube3Action”;
    - una Swipe sinistra sull’oggetto *CubeTre* causi su di esso la riproduzione di un effetto sonoro ed imposti la visibilità di *TorusTre* a non visibile.

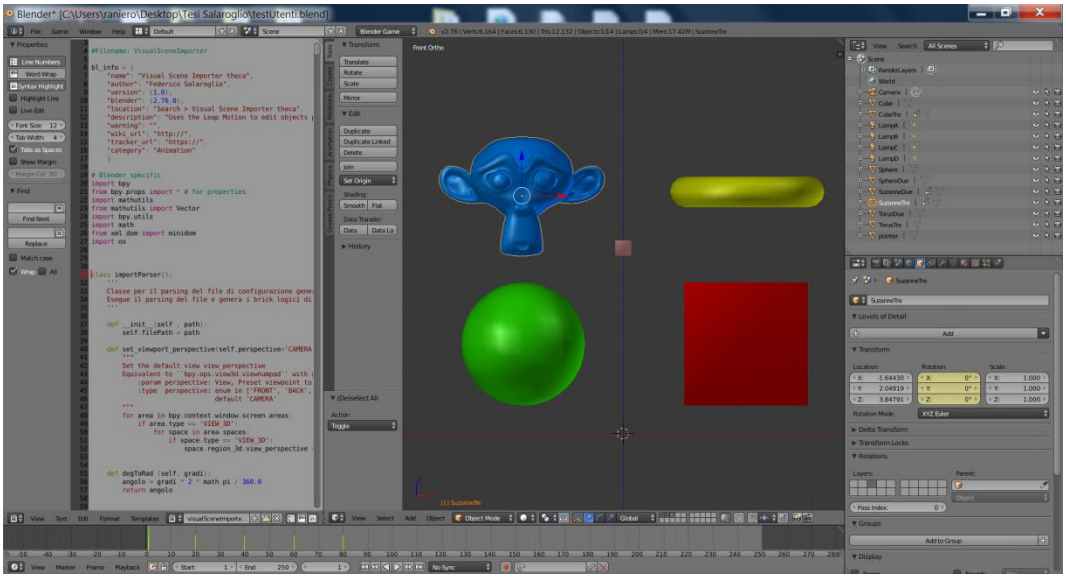


Figura 93. File Blender del test utenti di partenza.

Prima di chiedere ad altri soggetti di ricreare la presentazione ideata, si è provato a crearla. Per avere una visione generale del risultato che si otterrebbe, si ha realizzato il test progettato in ogni ambiente possibile: lo si ha creato nella nuova applicazione Visual Scene Editor e in quella precedente Leap Embedder, come anche direttamente all'interno di Blender attraverso il Logic Editor, distinguendo in quest'ultimo tra la versione equivalente alle sole interazioni richieste e le versioni complete di ogni componente necessario a un corretto funzionamento della simulazione via Leap Motion o con mouse e tastiera.

	Scene	**Oggetti**	**Mattoni logici**	**Tempo (min)**
Visual Scene Editor	4	13	15 pulsanti When/Do e 9 collegamenti tirati	2:30
Leap Embedder	3	24	30 blocchi gesture e action, mentre i collegamenti tra essi sono disegnati in automatico, però prima di creare un nuovo blocco è richiesto di selezionare a mano l'elemento a cui connetterlo	7
Blender – limitato a interazione richiesta	3	26	54 blocchi (sensori, controllori e attuatori) più 3 proprietà create in oggetti e 39 collegamenti creati tra di essi	14
Blender – completo per mouse e tastiera	3	28	181 blocchi più 34 proprietà create in oggetti e 138 collegamenti creati tra di essi	30
Blender – completo per Leap Motion	3	30	193 blocchi più 39 proprietà create in oggetti e 144 collegamenti creati tra di essi	35

Figura 94. Tabella comparativa dei componenti creati nei vari programmi.

Nella tabella in Figura 94 sono riportati, per ogni contesto, il numero di scene che è stato necessario creare, il numero di oggetti e quello dei mattoni logici per le interazioni richieste, indifferentemente dal fatto che questi siano pulsanti When e Do, blocchi gesture e action o sensori, controllori e attuatori.

Si sono distinte tre versioni per Blender: la prima è l'equivalente delle interazioni definite attraverso l'applicazione, mentre le altre due consistono nel test completo di ogni blocco logico necessario a funzionare, distinte tra versione per Leap Motion con teca e versione per tastiera e mouse con monitor, come risulterebbero attraverso l'importer che legge il file XML e comprendono molti più elementi, in quanto devono gestire anche quei componenti creati dall'importer stesso (come il funzionamento del puntatore assieme al rayEmitterObject per rilevare a cosa si sta puntando o il globalEmptyObject attraverso il quale passano tutti i Message di gesti rilevati). Dai valori riportati in tabella, si può notare come la nuova applicazione richieda una scena in più degli altri programmi, che corrisponde a Schema, ma riduce il numero di oggetti complessivo. I mattoni logici da definire risultano dimezzati rispetto al Leap Embedder e, anche se si devono creare a mano dei collegamenti, l'applicazione precedente disegnava le connessioni in automatico solo se prima di creare un nuovo blocco si selezionava a mano l'elemento a cui agganciarlo.

Relativamente ai tempi impiegati per realizzare la simulazione, con la nuova applicazione Visual Scene Editor si sono impiegati circa 2:30 minuti, mentre nel Leap Embedder circa 7 minuti. Nel Blender Game Engine si impiega circa 14 minuti, ma limitandosi a creare scene, oggetti e mattoni logici dell'interazione richiesta, senza tutti i componenti di fondo che permettono il riconoscimento gesti del Leap Motion o il rayEmitterObject che traccia il puntatore del mouse. Si segnala che in tutti i tempi indicati non si tiene conto del caricamento iniziale del file Blender (che nel Leap Embedder richiede più di un minuto) e del salvataggio finale del file XML.

Ritornando al test di usabilità proposto ai volontari, si è fatto ricorso a tre gruppi di utenti, con i quali si è affinata la procedura di valutazione.

In una prima fase si è proceduto a far eseguire il test a 14 utenti, ai quali si faceva loro presente in corso d'opera se avevano commesso un qualche errore, in modo che lo correggessero subito dopo. Per ogni persona si sono misurato i tempi impiegati per ottenere il risultato desiderato in modo corretto utilizzando il Leap Embedder e l'attuale Visual Scene Editor.

In tutti i casi si è ottenuto che con il Visual Scene Editor, il tempo impiegato era inferiore e risultava quasi sempre della metà circa, in riferimento al singolo utente. In Figura 95 sono riportati i tempi impiegati dagli utenti con le due versioni, dove si può notare come la nuova porti a ottenere perlopiù dei tempi inferiori alla precedente, mentre in Figura 96 sono mostrati i tempi di riferimento e i tempi medi tra tutti gli utenti (8:20 minuti con il Leap Embedder e 4:27 minuti con il Visual Scene Editor) con relativa deviazione standard.

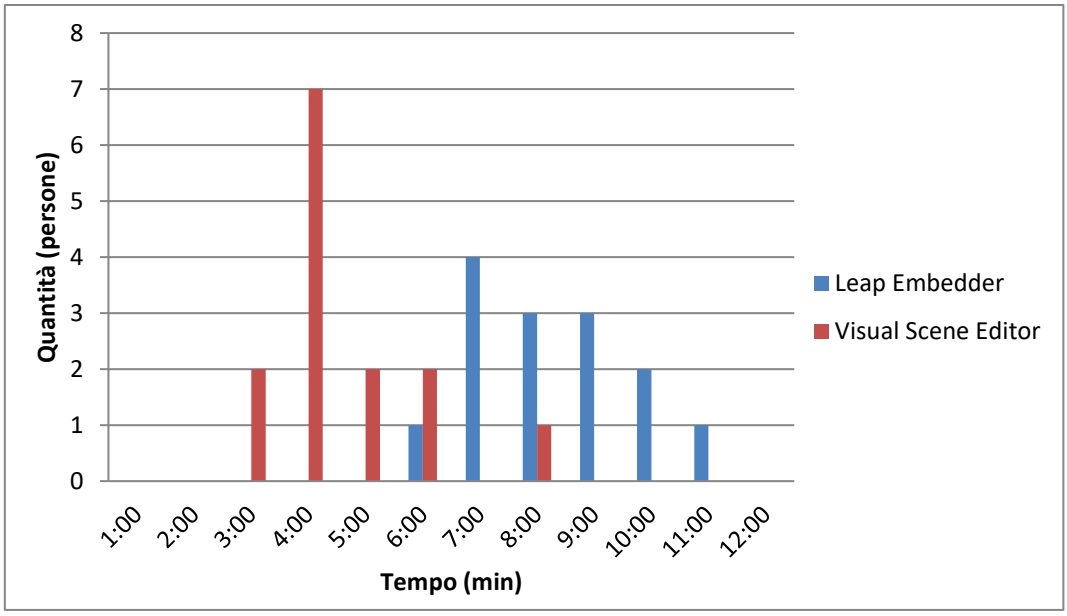


Figura 95. Diagramma a barre dei tempi impiegati dal primo gruppo di utenti.

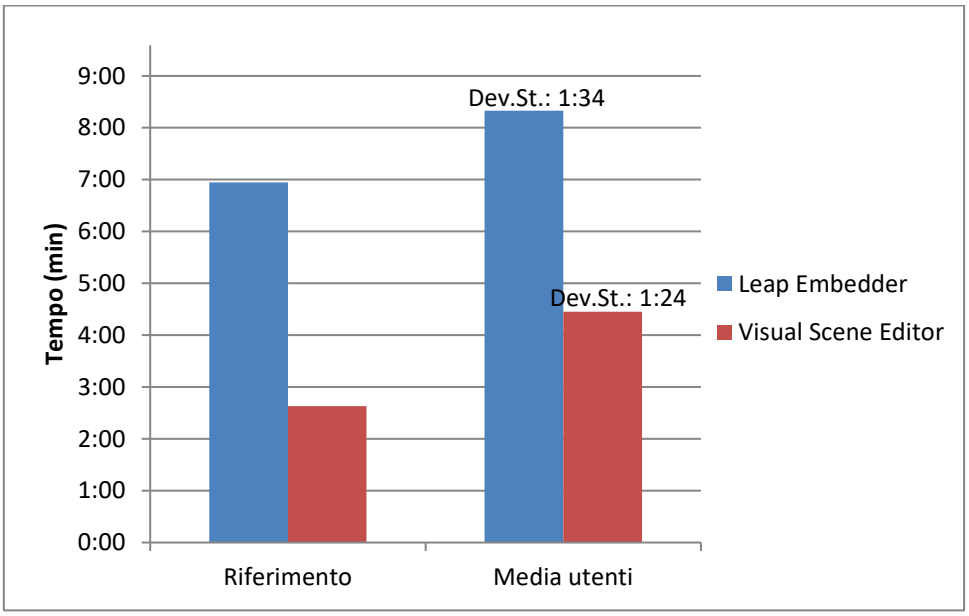


Figura 96. Diagramma a barre comparativo tra i tempi di riferimento e quelli del primo gruppo di utenti.

Comprendendo come questo test non fosse in realtà quello ideale per confrontare l'intuitività dei due programmi, si è proceduto a eseguire dei nuovi test su un nuovo campione di volontari.

Ad un secondo gruppo di 15 utenti si ha chiesto di svolgere il medesimo test, ma questa volta senza che venissero loro segnalati eventuali errori commessi. Al termine del lavoro si è misurato il tempo impiegato, informandoli solo alla fine se c'era almeno un errore e, in caso affermativo, concedendo altro tempo per fare in modo che lo trovassero e lo correggessero, registrando il tempo impiegato per arrivare

al risultato corretto. In questo caso, i dati raccolti comprendono un tempo di prima realizzazione e un secondo tempo richiesto a correggere eventuali errori commessi.

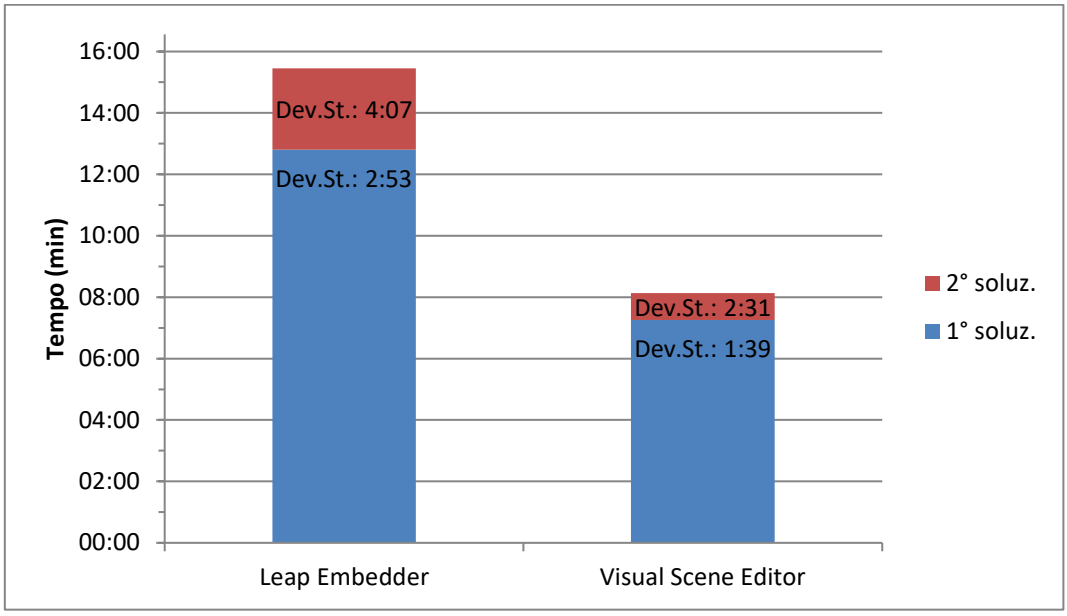


Figura 97. Diagramma a barre comparativo dei tempi medi impiegati dal secondo gruppo di utenti nelle due applicazioni.

In Figura 97 sono indicati i tempi nelle due applicazioni. Si può osservare come gli utenti abbiano impiegato più tempo, in generale, a completare il test senza alcuna segnalazione sugli errori eventualmente commessi, ma in ogni caso resta sempre evidente come con il Visual Scene Editor siano riusciti a completare il test in meno tempo rispetto al Leap Embedder.

Il terzo e ultimo gruppo di 16 utenti ha svolto il test nelle stesse condizioni del precedente, ma questa volta li si faceva partire alternatamente tra i due programmi, anziché sempre prima dal Leap Embedder come avvenuto nelle prime due sessioni di test, in modo da limitare l'impatto di fenomeni di learning.

Inoltre, in questa sessione si è fatto compilare a tutti i volontari un questionario di valutazione dell'esperienza, in modo da ottenere anche un riscontro soggettivo.

I soggetti coinvolti spaziano tra i 20 e i 30 anni, in entrambi i sessi e in media si sono dichiarate poco pratiche nell'uso di strumenti di grafica 3D e programmi di editing basati su nodi, mentre per nulla pratiche di linguaggi visuali di programmazione e strumenti per realizzare videogiochi o applicazioni grafiche interattive.

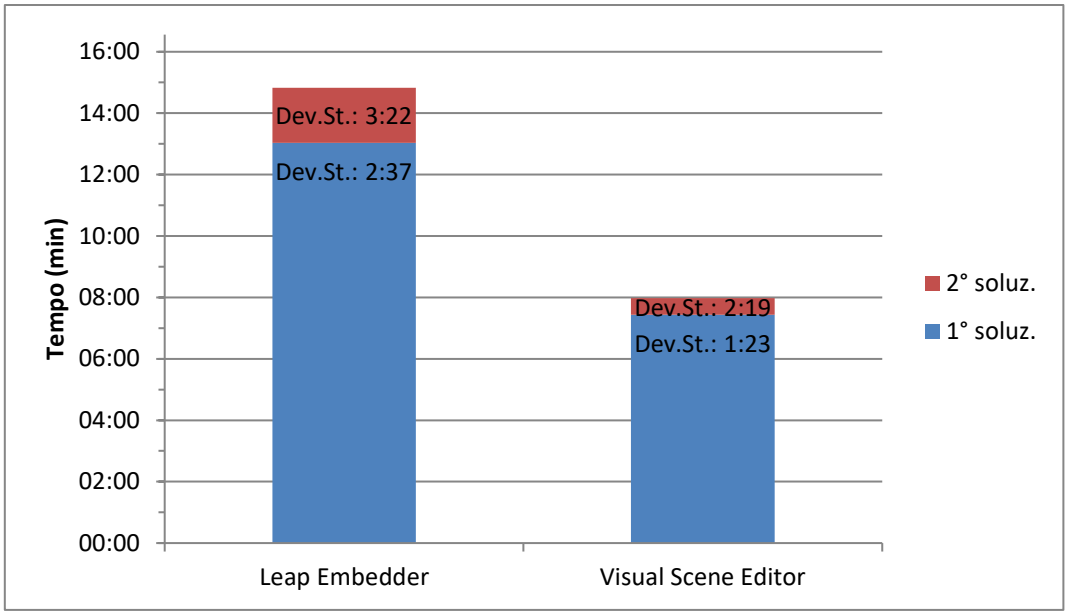


Figura 98. Diagramma a barre comparativo dei tempi medi impiegati dal terzo gruppo di utenti nelle due applicazioni.

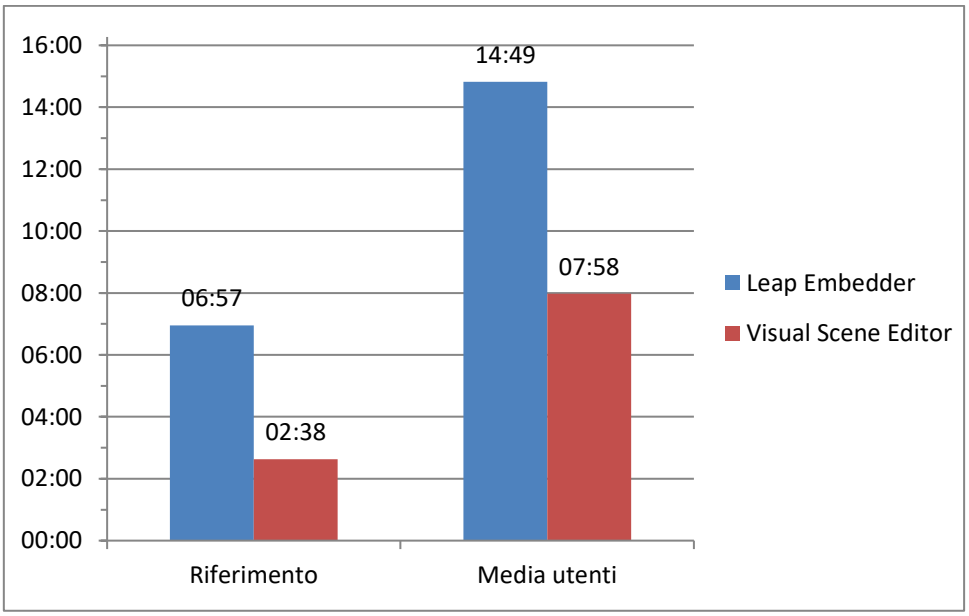


Figura 99. Diagramma a barre comparativo tra i tempi di riferimento e quelli del terzo gruppo di utenti.

Dai tempi riportati in Figura 98 e 99 si evince come con il Visual Scene Editor si sia mantenuta la tendenza a dimezzare quasi i tempi impiegati con il Leap Embedder, con un miglioramento percentuale pari al 45%.

Inoltre è interessante segnalare come si sia verificato spesso che un soggetto completasse il test senza commettere errori con il Visual Scene Editor, mentre utilizzando il Leap Embedder si sono riscontrate maggiori soluzioni inizialmente errate. Più precisamente su 16 persone si sono ottenuto 6 risultati

corretti al primo tentativo nel Leap Embedder, mentre ben 13 corretti nel Visual Scene Editor. Se nel Leap Embedder la quantità di errori commessi era mediamente pari a 3 o 4 per volta, nella nuova applicazione le rare volte che se ne rilevavano erano limitati a 1 o 2. Pertanto anche in questo senso si può affermare che la soluzione proposta in questo lavoro riesce a limitare la quantità di errori compiuti.
La causa di errore più frequente è legata all'uso dei Message nel Leap Embedder, che inizialmente risultava ostico da padroneggiare ed infatti è stato spesso segnalato tra i principali difetti di quell'applicazione. In seconda posizione erano frequenti errori nell'inserimento degli oggetti corretti nelle scene nel Leap Embedder, mentre nel Visual Scene Editor gli oggetti risultavano più chiari nelle scene e questo errore non capitava. Nel Leap Embedder la soluzione corretta al test proposto è una sola, composta da 30 blocchi logici. Nella nuova applicazione si può ottenere più di una soluzione valida; infatti la maggior parte degli utenti ha fatto ricorso a 15 blocchi logici, mentre un terzo di essi ne ha impiegati 16 (ed erano possibili anche soluzioni più articolate).
Gli utenti che hanno compilato il questionario hanno anche espresso una valutazione dell'interfaccia delle due applicazioni con un voto da 1 a 5 per i cinque attributi di usabilità di Nielsen: facilità di apprendimento, efficienza d'uso, facilità di ricordo, robustezza e soddisfazione [72]. In Figura 100 si nota come per ciascuno di questi aspetti si è ottenuto mediamente un punto in più con il Visual Scene Editor, che è stato considerato superiore al Leap Embedder.

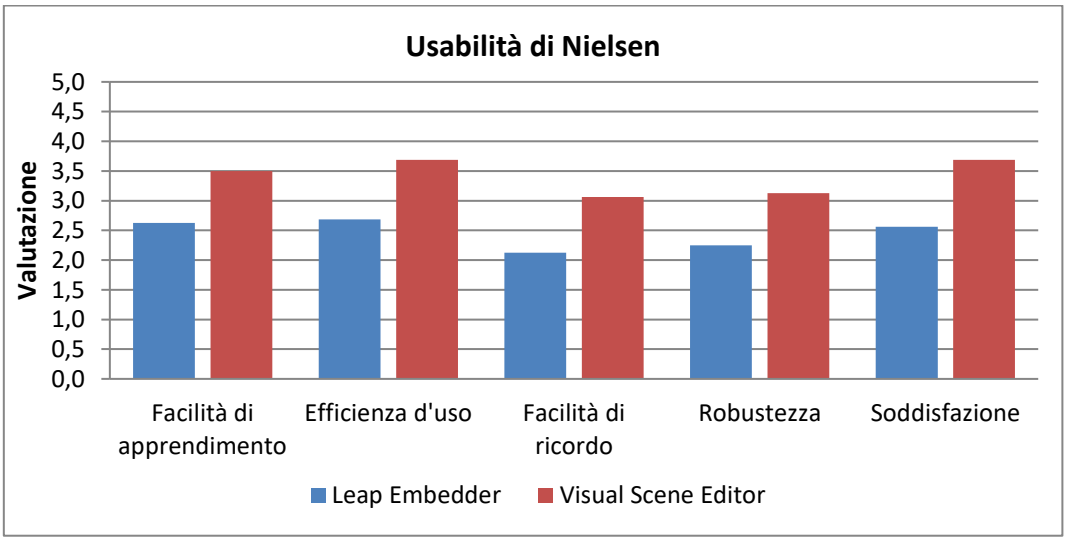


Figura 100. Diagramma a barre comparativo della valutazione ottenuta sugli attributi di usabilità di Nielsen nelle due applicazioni.

La superiorità del Visual Scene Editor è stata rafforzata anche dalla valutazione generale fornita dagli utenti, che hanno espresso una preferenza di utilizzo nei riguardi della nuova applicazione, come illustrato in Figura 101. Invece, quando si è chiesto loro quale delle due modalità preferissero per gestire le scene, gli oggetti e le relazioni e interazioni, i risultati mostrati sempre in Figura 101 evidenziano come le persone hanno chiaramente asserito di preferire il Visual Scene Editor per quanto riguarda la

gestione delle scene e delle relazioni e interazioni, mentre rispetto alla gestione degli oggetti in media gli utenti tendono a preferire sempre la nuova applicazione, ma in misura meno marcata. Questo è spiegabile dal fatto che quest'ultima è un'applicazione scene-centric, che fa risaltare le scene e le interazioni presenti, mentre il Leap Embedder è più object-centric, concentrando l'attenzione su un singolo oggetto alla volta.

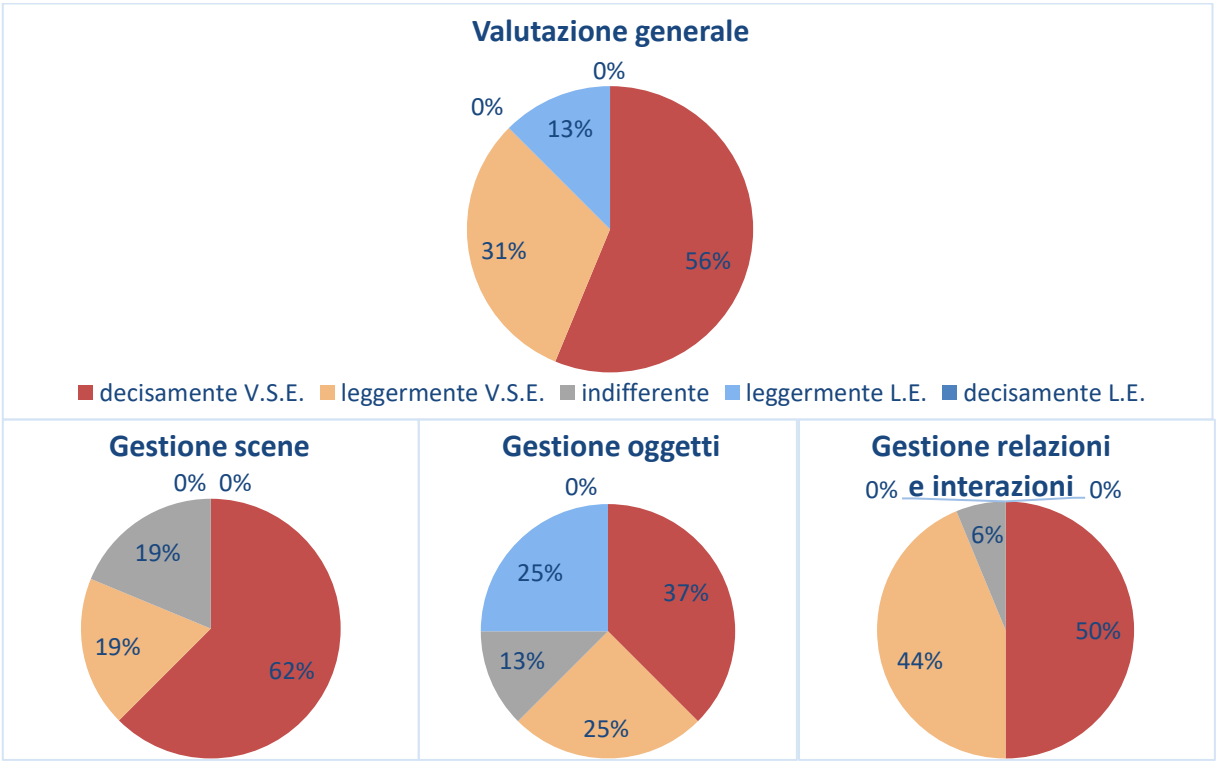


Figura 101. Diagrammi a torta sulla valutazione generale e la gestione di singoli aspetti da parte degli utenti.

Infine, si è chiesto alle persone di indicare uno o più pro e contro per entrambe le applicazioni e, come già affermato in precedenza, riguardo al Leap Embedder non sono mancati i difetti segnalati, perlopiù legati al concetto dei messaggi e una scarsa intuitività, che al contrario diventavano parte dei vantaggi per i quali preferire il Visual Scene Editor. I pochi difetti indicati per la nuova applicazione riguardavano invece aspetti estetici di icone e casi simili, che dopo la segnalazione si è provveduto a correggere.

In base a quanto evidenziato dai risultati soggettivi e oggettivi, si può concludere che la nuova applicazione sia riuscita nello scopo di migliorare quella già esistente, in quanto si sono ottenuti rilevanti riscontri positivi non solo in termini di tempi impiegati e numero di errori, ma anche relativamente all'intuitività e facilità di utilizzo. Anche gli utenti che hanno partecipato ai test erano concordi nell'asserire che la nuova versione risultava più semplice da utilizzare.

Capitolo 6

Conclusioni

In questo capitolo finale, si trarranno le conclusioni sul lavoro di tesi svolto, analizzando brevemente se il programma ottenuto, al netto dei risultati ottenuti dai test degli utenti, soddisfi quegli obiettivi che ci si era inizialmente posti di raggiungere. Al termine, si proveranno a ipotizzare quali potrebbero essere gli eventuali sviluppi futuri per espandere l'applicazione realizzata.

6.1 Conclusioni

Rispetto allo strumento di riferimento, il lavoro svolto ha portato ad ottenere un'applicazione di editing fortemente scene-centric, che pone maggiormente in risalto la struttura delle scene, consentendo all'utente di una avere una visione globale della simulazione più chiara e completa.
Si è visto come i test condotti con gli utenti abbiano confermato che la nuova applicazione risulti significativamente più intuitiva, consentendo di ridurre sensibilmente i tempi di sviluppo oltre che le possibilità di errore. I concetti poco immediati (come, ad esempio, quello di "messaggio") ereditati dal Logic Editor di Blender sono stati nascosti all'utente, che può operare ad un livello di astrazione più elevato.
In quanto a versatilità dall'applicazione, sono state mantenute le funzioni principali del precedente strumento, migliorando alcuni aspetti e potenziandone altri con nuove funzionalità, come illustrato in fase di analisi comparativa tra le due versioni, concedendo così maggiore libertà agli utilizzatori. Dunque, anche sotto questo aspetto è possibile trovare dei vantaggi nell'utilizzo dello strumento proposto in questo lavoro.
Infine, l'interfaccia è stata progettata per essere nell'eventualità facilmente espandibile in futuro di nuove tipologie di blocchi logici When e Do, permettendo di riadattarla ad altri ambiti di utilizzo o ad altri dispositivi di interazione.

6.2 Possibili sviluppi futuri

Tra i possibili sviluppi futuri il principale si può citare l'estensione delle interazioni disponibili: sia i When sia i Do sono stati progettati fin da subito in modo da risultare facilmente estendibili, attraverso l'aggiunta di un ulteriore tab alla finestra di selezione adatto a contenere nuove tipologie di interazione.
Per i When si potrebbe ad esempio completare l'implementazione del tab Proximity, per gestire gli eventi spaziali come le collisioni, che non si era terminato in quanto non ritenuto fondamentale in un primo momento, oppure aggiungere altre funzioni che si riterranno utili in futuro.
Inoltre si potrebbe potenziare il set di gesti riconoscibili dal Leap Motion, attualmente limitato a quelli rilevati nativamente dal framework di sviluppo, ricorrendo a un modulo che utilizzi ad esempio algoritmi di classificazione dedicati.
Se invece si volesse utilizzare l'applicazione in combinazione con un differente dispositivo di tracciamento del movimento, come per esempio il Microsoft Kinect, si potrebbe semplicemente sostituire i nomi del tab gesture con i nuovi gesti, o più in generale input, che si desidera riconoscere, limitandosi a riadattare l'importer. In alternativa si potrebbe aggiungere un nuovo tab per permettere un approccio multi-modale, che oltre ai gesti implementati consenta anche un'interazione basata, ad esempio, su comandi vocali.
Un altro interessante ampliamento che si potrebbe valutare di implementare consiste nell'affiancare un ambiente di esecuzione semplificato, dove simulare le interazioni delle scene create, così gli utilizzatori possono eseguire i loro programmi realizzati per vedere subito e comodamente un'anteprima dei risultati, direttamente all'interno del Visual Scene Editor.
Infine si segnala che, se lo si desidera, il progetto generato nell'applicazione può essere importato anche in altri game engine, per avere funzionalità e logica differenti da quelle offerte dal BGE, per esempio progettando un modulo di importazione apposito per Unity 3D da affiancare a quello esistente per Blender.

Bibliografia

[1] Interazione uomo-macchina – *en.wikipedia.org/wiki/Human%E2%80%93computer_interaction*, 16 ottobre 2017

[2] Gamberini, Chittaro, Paternò, (2012), "*Human-Computer Interaction - I fondamenti dell'interazione tra persone e tecnologie*"

[3] Stuart K. Card, Allen Newell, Thomas P. Moran, (1983), "*The Psychology of Human-Computer Interaction*"

[4] Kieras, Polson, (1985), "*The Acquisition and Performance of Text-Editing Skill: A Cognitive Complexity Analysis*"

[5] Luigi Maggio, (2014), "*Progettazione e sviluppo di un framework per l'integrazione di dispositivi di interazione naturale con sistemi di proiezione olografica*", Politecnico di Torino

[6] Linguaggi di programmazione visuali – *en.wikipedia.org/wiki/Visual_programming_language*, 16 ottobre 2017

[7] Panoramica sui VPL – *www.craft.ai/blog/the-maturity-of-visual-programming/*, 16 ottobre 2017

[8] Limite di Deutsch – *en.wikipedia.org/wiki/Deutsch_limit*, 16 ottobre 2017

[9] VPL lista – *blog.interfacevision.com/design/design-visual-progarmming-languages-snapshots/*, 16 ottobre 2017

[10] Linguaggi di programmazione educativi – *en.wikipedia.org/wiki/List_of_educational_programming_languages*, 16 ottobre 2017

[11] VPL sintesi – *constructingkids.com/2013/05/15/vpl/*, 16 ottobre 2017

[12] Blockly – *developers.google.com/blockly/*, 16 ottobre 2017

[13] EToys – *www.squeakland.org/*, 16 ottobre 2017

[14] HopScotch – *www.gethopscotch.com/*, 16 ottobre 2017

[15] LEGO Mindstorms – *www.lego.com/en-gb/mindstorms*, 16 ottobre 2017

[16] miniBloq – *blog.minibloq.org/*, 16 ottobre 2017

[17] Scratch – *scratch.mit.edu/*, 16 ottobre 2017

[18] ToonTalk – *en.wikipedia.org/wiki/ToonTalk*, 16 ottobre 2017

[19] TouchDevelop – *www.touchdevelop.com/*, 16 ottobre 2017

[20] Snap! – *snap.berkeley.edu/*, 16 ottobre 2017

[21] Flowgorithm – *www.flowgorithm.org/*, 16 ottobre 2017

[22] Flow Hub – *flowhub.io/*, 16 ottobre 2017

[23] OpenDX – *www.opendx.org/*, 16 ottobre 2017

[24] SQL Server Integration Services – *www.microsoft.com/it-IT/sql-server/sql-server-2017*, 16 ottobre 2017

[25] Visual Logic – *www.visuallogic.org/*, 16 ottobre 2017

[26] AudioMulch – *www.audiomulch.com/*, 16 ottobre 2017

[27] Audulus – *audulus.com/*, 16 ottobre 2017

[28] Blender – *www.blender.org/*, 16 ottobre 2017

[29] CryEngine – *www.cryengine.com/*, 16 ottobre 2017

[30] MeVisLab – *www.mevislab.de/*, 16 ottobre 2017

[31] Nuke – *www.foundry.com/products/nuke*, 16 ottobre 2017

[32] Simulink – *it.mathworks.com/products/simulink.html*, 16 ottobre 2017

[33] Unity 3D – *unity3d.com*, 16 ottobre 2017

[34] NodeCanvas – *nodecanvas.paradoxnotion.com/*, 16 ottobre 2017

[35] RAIN, Rival Theory – *legacy.rivaltheory.com/rain/*, 16 ottobre 2017

[36] AngryAnt Behave – *angryant.com/behave/*, 16 ottobre 2017

[37] craft ai – *www.craft.ai/*, 16 ottobre 2017

[38] Behavior3 – *behavior3.com/*, 16 ottobre 2017

[39] AgentCubes / AgentSheets – *www.agentsheets.com/agentcubes/*, 16 ottobre 2017

[40] GameSalad – *gamesalad.com/*, 16 ottobre 2017

[41] Kodu – *www.kodugamelab.com/*, 16 ottobre 2017

[42] Project Spark – *en.wikipedia.org/wiki/Project_Spark*, 16 ottobre 2017

[43] Clickteam Fusion – *www.clickteam.com/clickteam-fusion-2-5*, 16 ottobre 2017

[44] Scratch – *en.wikipedia.org/wiki/Scratch_(programming_language)*, 16 ottobre 2017

[45] ScratchJr – *www.scratchjr.org/*, 16 ottobre 2017

[46] J. Maloney, M. Resnick, N. Rusk, B. Silverman, E. Eastmond, (2010), "*The Scratch Programming Language and Environment*"

[47] KODU Game Lab – *en.wikipedia.org/wiki/Kodu_Game_Lab*, 16 ottobre 2017

[48] Download Kodu Game Lab – *www.microsoft.com/en-us/download/details.aspx?id=10056*, 16 ottobre 2017

[49] Kodu – *www.microsoft.com/en-us/research/project/kodu/*, 16 ottobre 2017

[50] Project Spark wiki fandom – *projectspark.wikia.com/wiki/Project_Spark_Wiki*, 16 ottobre 2017

[51] Blender wiki – *en.wikipedia.org/wiki/Blender_(software)*, 16 ottobre 2017

[52] Ton Roosendaal – *en.wikipedia.org/wiki/Ton_Roosendaal*, 16 ottobre 2017

[53] Blender documentazione – *docs.blender.org/manual/en/dev/*, 16 ottobre 2017

[54] Laura Venturi, (2013), "*Exhibit interattivo di oggetti tridimensionali*", Politecnico di Torino

[55] Leap Motion – *blog.leapmotion.com*, 16 ottobre 2017

[56] Leap Motion wiki fandom – *leap-motion.wikia.com/wiki/Leap_Motion_Wiki*, 16 ottobre 2017

[57] Leap Motion documentazione – *developer.leapmotion.com*, 16 ottobre 2017

[58] JavaFX – *en.wikipedia.org/wiki/JavaFX*, 16 ottobre 2017

[59] JavaFX documentazione e tutorial – *docs.oracle.com/javase/8/javase-clienttechnologies.htm*, 16 ottobre 2017

[60] JavaFX API documentazione – *docs.oracle.com/javase/8/javafx/api/toc.htm*, 16 ottobre 2017

[61] JavaFX Home – *www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html*, 16 ottobre 2017

[62] Gluon Scene Builder – *gluonhq.com/products/scene-builder/*, 16 ottobre 2017

[63] International Standards Organization, (1994), "*Ergonomic requirements for office work with visual display terminals.*", Part 11: Guidance on usability (ISO DIS 9241-11)

[64] Roberto Polillo, (2010), "*Facile da usare*", Apogeo

[65] Jakob Nielsen, (1993), "*Usability Engineering*", Academic Press

[66] Usabilità – *en.wikipedia.org/wiki/Usability*, 16 ottobre 2017

[67] Maria Francesca Costabile, Carmelo Ardito, Rosa Lanzilotti, Grazia Minardi, Antonio Piccinno, Carlo Dell'Aquila, (2005), "*Linee Guida per il Progetto e la Valutazione di Sistemi Interattivi Visuali Usabili*"

[68] Antonella De Angeli, (1997), "*Valutare i sistemi flessibili: un approccio globale alla HCI*", Università degli Studi di Trieste

[69] Model-View-Controller – *en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller*, 16 ottobre 2017

[70] JAXB – *en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding*, 16 ottobre 2017

[71] Busto di Nefertiti – *en.wikipedia.org/wiki/Nefertiti_Bust*, 16 ottobre 2017

[72] Usabilità di Nielsen – *en.wikipedia.org/wiki/Jakob_Nielsen_(usability_consultant)*, 16 ottobre 2017

Ringraziamenti

Nel concludere questa tesi, non posso fare a meno di ringraziare le persone che mi hanno accompagnato. A partire dai miei genitori, che mi hanno sostenuto permettendomi di portare a compimento questo lavoro e terminare il lungo percorso di studi intrapreso.
Un ringraziamento particolare va ai miei relatori, i professori Fabrizio Lamberti e Andrea Sanna, per la loro disponibilità durante tutto lo svolgimento del progetto e i loro preziosi consigli.
A tutti gli amici che mi sono stati accanto durante la stesura di questa tesi, condividendo giornate di studio e svago. Grazie per la loro disponibilità anche a tutti i numerosi volontari che si sono prestati a svolgere i test di valutazione conclusivi.